UNIVERSIDAD CARLOS III

Escuela Politécnica Superior

DEPARTAMENTO DE INGENIERIA DE TELEMÁTICA

DESARROLLO DE UNA APLICACIÓN DE CONTROL DE ACCESO Y SISTEMAS DE IDENTIFICACIÓN MEDIANTE LA TECNOLOGÍA NFC

Autor: Elena Vicente García

Tutor: Mario Muñoz Organero

Leganés, Junio 2011

Agradecimientos

Quiero darle las gracias a Mario por su apoyo y paciencia en el transcurso del desarrollo del proyecto y por brindarme la oportunidad de poder trabajar para la empresa Gamma, a quien también estoy agradecida, por poder realizar un proyecto con una tecnología innovadora.

Quiero agradecer a todas esas personas que he ido conociendo a lo largo de la carrera, con los que he compartido momentos geniales y divertidos, y a los que tengo muy presentes, Luis, Mercedes y especialmente a Elena, que ha sabido estar a mi lado en momentos difíciles ofreciéndome su ayuda incondicional. A mis compañeros de clase y ahora amigos, por todo lo que he aprendido de cada uno de ellos, agradeceros todos esos momentos que no olvidaré, y que me hacen sentir bien, por ser tan distintos, por ayudarme tanto durante la carrera, por marcarme de una forma especial cada uno de vosotros, Adal, Carlitos, Chemi, Dieguete, Jesus, Lisardo, Miguel, Pablete, Rober, Samu, Virgi , David y Aida.

Quiero agradecer a mis padres, Esther y Vicente, su apoyo y paciencia durante toda la carrera, y la ilusión compartida en todo momento durante el desarrollo del proyecto. Por haber sacrificado tanto para que pudiese llegar hasta aquí.

Gracias a mis hermanas, Noelia y Virginia, por todas las cosas que os debo y que soy incapaz de resumir. Por estar presentes en muchos momentos de mi vida. Junto a Héctor quiero agradeceros el ser tan comprensivos, pacientes, a animarme y a no desesperar!

A Adrian, una persona muy especial, por compartir tanto conmigo y estar a mi lado durante todos estos años, por formar ya parte de mi vida. Gracias por darme tanto, por ayudarme, por tus abrazos, por tus sonrisas, y especialmente por saltar conmigo en los charcos.

A todos vosotros Gracias!

Resumen

La tecnología NFC ha adquirido en este último año una presencia importante en nuevos proyectos, algunos de ellos a gran escala. Está diseñada para que la interacción entre los dispositivos sea de forma intuitiva, simple y segura. Además promete ser útil y atractiva al usuario, contando con la ventaja de la interoperabilidad con otras tecnologías inalámbricas existentes.

Ha supuesto una revolución al ofrecer los servicios de las tarjetas inteligentes y las ventajas de las tecnologías inalámbricas de corto alcance. Posee una gran proyección de futuro, debido al gran potencial que presenta a la hora de desarrollar nuevas aplicaciones, lo que indica que se convertirá en indispensable dentro de pocos años. Cambiará nuestra manera de relacionarnos con el entorno al convertir nuestro dispositivo móvil en monedero, llavero o mando a distancia, entre otras muchas opciones.

Además de esto, hoy en día los dispositivos móviles se han convertido en elementos importantes e imprescindibles en nuestra vida diaria.

Con este proyecto nos acercamos a la tecnología NFC (*Near Field Communication*) integrada en los dispositivos móviles, aprovechando las posibilidades que nos ofrece. A través de ello, se ha desarrollado una aplicación que permite implementar sistemas de identificación y control de acceso.

Abstract

Last year NFC technology had acquired a significant presence in new projects, some of at large scale. It has been designed with special focus into the interaction between devices, resulting an intuitive, simple and secure process. It also promises to be useful and attractive to the user, with the advantage of interoperability with other existing wireless technologies.

NFC has brought a revolution offering smart card services and benefits of short-range wireless technologies. It has a bright future, due to the great potential it offers to new development applications. This is an indication that it will become a very important piece into mobile applications in a matter of only a few years. They can change the way we interact with the environment by turning our mobile devices into a wallet, key ring or even a remote control, among other options. After all, today's mobile devices have become an important and essential device in our daily lives.

With this new project, we present the integration of NFC (Near Field Communication) on our mobile devices, taking advantage of the possibilities it has to offer. With this new technology we have developed an application that allows us to implement systems for authentication and accesscontrol.

ÍNDICE GENERAL

Índice de Figuras

Índice de Tablas

CAPÍTULO 1
INTRODUCCIÓN

Este capítulo ofrece una breve introducción a la tecnología empleada, así como una descripción de las razones que motivaron la realización de este proyecto. Posteriormente se presentan los objetivos a alcanzar.

1.1. MOTIVACIÓN

La tecnología ha buscado desde sus comienzos facilitar el día a día de las personas, mejorar la interacción de estas con el resto y con el medio que les rodea y ofrecer otras alternativas en ocio y tiempo libre. Uno de los avances que brinda todo esto es el dispositivo móvil. Se creó para satisfacer la necesidad de comunicación, ofreciendo movilidad a las personas.

Actualmente estamos presenciando una revolución tecnológica, los nuevos dispositivos móviles ofrecen multitud de aplicaciones que permiten tener en un mismo dispositivo de tamaño reducido infinidad de funcionalidades. Su crecimiento acelerado ha incentivado que se convierta en una herramienta de uso diario, necesario e indispensable.

La tecnología NFC, está pensada para ser integrada en los dispositivos móviles, por lo que es una tecnología con **gran potencial, alcance y disponibilidad**. De esta forma los usuarios podrán tener acceso a un número mayor de servicios a través de su dispositivo móvil y en cualquier momento. Esta es la ventaja real de NFC, que es una tecnología que se encuentra en los teléfonos móviles, con gran pantalla y con conectividad con la red.

Otra característica que hace que su uso sea muy interesante, es la **compatibilidad con otras tecnologías inalámbricas** ya existentes. Por ello las empresas han mostrado gran interés en esta tecnología, invirtiendo en proyectos piloto y probando su desenvoltura con los servicios de proximidad. [1]

Es **sencilla de utilizar**, intuitiva y segura. Ya que sólo requiere que dos dispositivos se toquen con el fin de comunicarse. Supone un fácil acceso a servicios y contenidos ofrecidos por objetos físicos y otros dispositivos. Al estar integrada en un dispositivo móvil no necesita configuración por parte del usuario, por lo que la comodidad para éste está asegurada.

Es una **tecnología segura,** ya que para activar cualquier servicio o intercambiar información el usuario tendrá que hacerlo manualmente, o bien juntar dos dispositivos móviles, confirmando nuevamente el usuario cada acción. Además, como está integrada en un dispositivo móvil podemos incluir varios niveles de seguridad.

Todas estas cualidades confirman que es una tecnología con futuro, y que está adquiriendo gran importancia, además de popularizarse entre los usuarios.

Por todo ello, con la realización de este proyecto, se intenta dar a conocer de una forma más cercana el funcionamiento de la tecnología NFC, implementando un entorno real que permita sacar provecho de sus características y ventajas para la satisfacción del usuario.

1.2. OBJETIVOS

El objetivo principal de este proyecto es diseñar e implementar un conjunto de aplicaciones haciendo uso de la telefonía móvil y la tecnología NFC integradas en el sector turístico, que permita el desarrollo de nuevas aplicaciones.

El proyecto permite el control de acceso a las diferentes zonas que puede tener un hotel, como habitaciones, recepción, office, cocina o despachos. Además, cada empleado contará con un dispositivo móvil provisto de NFC, que contendrá los datos personales, para poder acceder a ellos cuando se precise, permitiendo la identificación de cada uno de los empleados.

Se implementará también la posibilidad de seguir las rutas seguidas por los guardias de seguridad en sus respectivos turnos (zona en la que ha estado y las horas a las que ha entrado y salido cada día).

Los datos personales de cada empleado y las zonas a las que cada uno tiene acceso se gestionan mediante una base de datos que contendrá toda la información, y una página web, desde donde podremos modificar estos datos de una forma fácil y asequible.

Para todo ello se utilizarán los dispositivos móviles NOKIA 6131, provistos de la tecnología NFC y Bluetooth. Simularemos las chip de los teléfonos donde se almacena la información de los usuarios mediante tarjetas Mifare. Implementaremos una BBDD donde almacenaremos la información referente a los empleados (trabajadores del hotel), y una página Web, que permita manejar estos datos de una forma intuitiva y fácil.

Objetivos a cumplir:

- Analizar y estudiar las tecnologías inalámbricas, Bluetooth y, más en detalle, NFC.

- Estudiar la forma de guardar de forma segura la información en las tarjetas Mifare mediante la tecnología NFC.

- Estudiar el protocolo NFCIP para el intercambio de datos entre los distintos dispositivos.

- Desarrollar e implementar una aplicación que me permita tener un control de acceso a las diferentes zonas e identificación de los empleados mediante la combinación de las tecnologías Bluetooth, NFC, NFCIP, MySQL.

- Desarrollar la aplicación web que maneje de forma sencilla los datos de la BBDD, gestione el control de acceso a las zonas, y tenga un seguimiento de las rutas de los vigilantes de seguridad.

- Desarrollar una batería de pruebas que me permita testear y comprobar el correcto funcionamiento de esta herramienta.

- Estudiar las posibles líneas de futuro que podrían desarrollarse para mejorar esta herramienta.

1.3. CONTENIDO DE LA MEMORIA

La memoria ha sido organizada de tal forma que se muestran los pasos que se han seguido para el desarrollo del presente proyecto.

En primer lugar, hay un proceso de documentación, el estado del arte, en el que se analizarán las tecnologías que se van a utilizar en el proyecto, con el fin de usar las técnicas más apropiadas para el mismo. Estudiaremos a fondo las tecnologías NFC y Bluetooth.

En el tercer capítulo, se explicarán las APIs empleadas en la aplicación, referentes a las tecnologías NFC y Bluetooth.

Se detalla, en el cuarto capítulo, la descripción funcional de todo el sistema, en concreto se centra en los objetivos, la funcionalidad, los requisitos y su diseño.

Una vez explicado lo que se quiere hacer y cómo se quiere realizar, se explicarán los detalles de la implementación, en el capitulo quinto.

Realizaremos una batería de pruebas para comprobar el correcto funcionamiento del proyecto, en el sexto capítulo.

En el capítulo séptimo evaluaremos los resultados obtenidos en el capitulo anterior sacando conclusiones y exponiendo propuestas para líneas futuras del proyecto.

Se presenta el presupuesto total del proyecto en el octavo capítulo.

Por último, el Anexo A, recoge la guía de instalación del software para poder instalar y desplegar todo el escenario con el que se ha desarrollado este proyecto, y el Anexo B, recoge el manual de usuario para poner en funcionamiento el sistema completo.

CAPÍTULO 2

ESTADO DEL ARTE

En este capítulo se hace un repaso de las tecnologías utilizadas en este proyecto. En primer lugar se presentarán las tecnologías inalámbricas utilizadas, NFC y Bluetooth.

Explicaremos la elección del servidor Web para alojar la BBDD, y las herramientas empleadas para la aplicación Web.

A continuación se presenta el lenguaje de programación Java, en concreto la plataforma J2ME, utilizada para el desarrollo de aplicaciones móviles.

Se detalla al final del capítulo la estructura de una tarjeta *Mifare*.

2.1. NEAR FIELD COMMUNICATION

2.1.1. INTRODUCCIÓN

NFC, *Near Field Communication*, es una tecnología inalámbrica de interconexión de dispositivos de corto alcance que surge de combinar la tecnología RFID y la tecnología de interconexión de *smart cards*. Permite realizar una comunicación simple, segura e intuitiva entre dispositivos.

NFC aparece como un progreso en la convergencia de aplicaciones dentro del teléfono móvil, ya que pretende crear una forma de comunicación, elaboración de pagos y almacenamiento de datos mucho más segura para los dispositivos electrónicos móviles. Permite interconectar la gran variedad de dispositivos que nos rodean, cada uno con una multifuncionalidad.

La principal característica que hace que la tecnología NFC sea interesante y atractiva, es la compatibilidad con otras tecnologías inalámbricas existentes como Bluetooth[1], RFID[2] y Wifi[3]. Aunque, a diferencia de estas tecnologías no está pensada para ser utilizada en una transmisión masiva de datos. Sin embargo, encontramos su punto fuerte en la velocidad de comunicación, ya que está pensada para un intercambio rápido de unos pocos bits de información. Por tanto podemos decir que es una comunicación casi instantánea, sin necesidad de emparejamiento previo.

A continuación trataremos de una forma más detallada esta tecnología, para entender lo que se puede lograr con ella, su funcionamiento y su uso en la actualidad.

2.1.2. PRINCIPALES CARACTERÍSTICAS

NFC es una tecnología de comunicación inalámbrica de corto alcance (los dispositivos deben estar entre sí aproximadamente a 10cm) que permite el intercambio bidireccional de datos.

[1] Bluetooth: *http://www.bluetooth.com/Pages/About-the-Technology.aspx*
[2] RFID: *http://es.wikipedia.org/wiki/RFID*
[3] WIFI: *http://es.wikipedia.org/wiki/Wi-Fi*

Se trata de una tecnología inalámbrica que surge con la idea de crear un nuevo protocolo que sea compatible con las tecnologías existentes de corto alcance, por ello es una extensión simple del estándar ISO/IEC 14443 de tarjetas de proximidad (etiquetas RFID), combinando la interfaz de una tarjeta inteligente y de un lector dentro de un mismo dispositivo.

NFC añade a la tecnología RFID la comunicación entre dos dispositivos activos. Y se diferencia de ésta básicamente por dos motivos:

1. NFC provee al mismo tiempo comunicación P2P (*peer to peer*), lo que permite a dos dispositivos interconectarse; mientras que los protocolos de las *contactless card* y *smart cards* solo soportan la comunicación entre dispositivos con energía y *tags* pasivos. Por lo que NFC se convierte en una tecnología mucho más interesante.
2. NFC no puede ser activado remotamente por accidente o involuntariamente. El teléfono obliga a que deba existir un acercamiento entre dispositivos antes de iniciar una comunicación.

Podemos decir por lo tanto que NFC es una evolución mejorada de las tecnologías existentes al servicio de los usuarios finales.

Figura 2.1: Evolución NFC

Comienza a desarrollarse conjuntamente entre Philips y Sony a finales de 2002, con el fin de crear una herramienta para las comunicaciones sin contacto (*contactless communications*). Crearon el protocolo NFC, para el desarrollo de un sistema compatible con las tecnologías *contactless* propietarias existentes en el mercado: *Felica* por parte de Sony y *Mifare* por parte de Philips. [2]

A finales de 2003 se aprueba como el estándar ISO 18092. Y en 2004, Philips, Nokia y Sony fundaron el **NFC Forum**[4], encargado del desarrollo de las especificaciones, garantizando la interoperabilidad entre los dispositivos NFC y servicios. NFC Forum introdujo la arquitectura estandarizada de la tecnología, las especificaciones iniciales y los formatos de *tags* para dispositivos compatibles con NFC.

[4] NFC FORUM: *http://www.nfc-forum.org/home*

Entre ellos se encuentra NDEF (*Data Exchange Format*), y tres especificaciones iniciales para RTD (*Record Type Definition*) para posters inteligentes, texto y aplicaciones de lectura de recursos de internet.

Fabricantes, desarrolladores de aplicaciones, instituciones de servicios financieros y otros han estado cooperando para promover el uso de la tecnología NFC en la electrónica de consumo, dispositivos móviles y PC.

2.1.2.1. ESPECIFICACIONES TÉCNICAS

Dentro de la forma de trabajo de NFC, los dispositivos generan una onda de radio de baja frecuencia operando a 13,56 MHz, disponible globalmente sin restricción y sin necesidad de licencia para su uso. Esta tecnología usa circuitos inductivos emparejados que pueden intercambiar un flujo de datos cuando están próximos entre sí. El acoplamiento inductivo funciona solo para distancias cortas. El uso de este tipo de acoplamiento es la principal diferencia entre NFC y otras tecnologías inalámbricas como Bluetooth y Wifi, que trabajan a una distancia máxima 10 y 100 metros respectivamente.

La comunicación es half-duplex (ambos sentidos, pero no simultáneamente), ya que se emplea una única portadora. Además los dispositivos NFC pueden verificar el campo de Radio Frecuencia y detectar una colisión si la señal recibida no coincide con la señal transmitida. [3]

Anteriormente dijimos que NFC no es una tecnología pensada para la transmisión masiva de datos, porque la tasa de transferencia de datos máxima aproximada es de 424 Kbps. En comparación con otras tecnologías como Bluetooth con una tasa de transferencia de 3 Mbps, o Wifi con 54 Mbps.

Las tarjetas (Tags) pueden insertarse en dispositivos como teléfonos móviles, PDAs, tarjetas de pago o tickets.

Existen tres configuraciones diferentes para utilizar NFC, lo que hace que sea más adaptable y eficiente que otras tecnologías:

- Emulación de tarjetas: El dispositivo NFC se comporta como una contactless card. Se puede utilizar las características de seguridad avanzadas del elemento seguro, siendo útil para el manejo de sistemas de pagos basados en diferentes métodos o controles de acceso.

- Modo Reader: Esta forma es una de las más comunes y utilizadas hoy en día, en donde el dispositivo NFC se encuentra en modo activo y lee un tag RFID pasivo; como lo es por ejemplo de lectura y almacenamiento de una dirección web o cupones publicitarios.
- Modalidad P2P: Dos dispositivos NFC se comunican entre si para el intercambio de información. La comunicación NFC se podría utilizar para establecer parámetros de una comunicación inalámbrica como Bluetooth o Wifi.

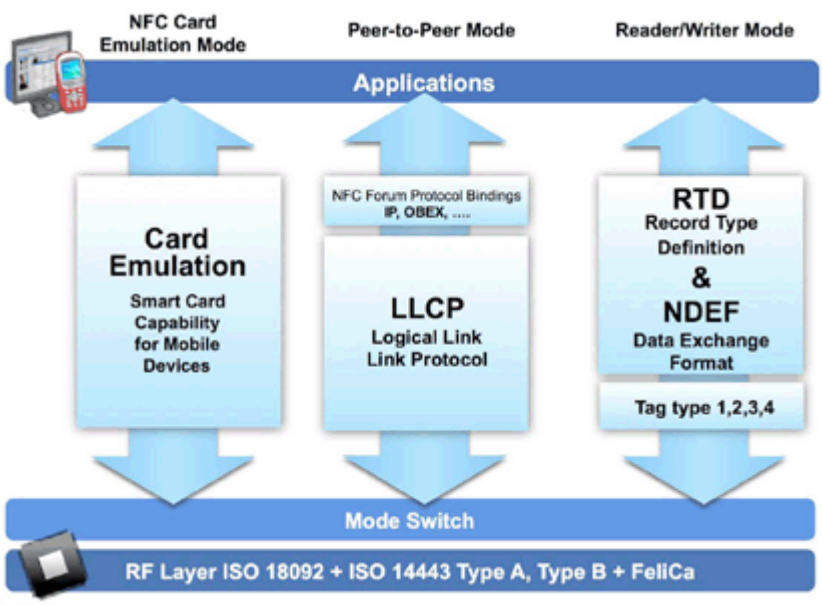


Figura 2.2: Arquitectura NFC [4]

2.1.2.2. ESTÁNDAR DE LA COMUNICACIÓN

NFC Forum ha definido un formato de datos común para que los dispositivos y las etiquetas NFC puedan compartir información entre sí.

NDEF (*NFC Data Exchange Format*) es un formato ligero de mensaje, que se utiliza para guardar y transportar diferentes tipos de elementos. Sólo especifica la estructura del formato, y éste es el mismo tanto para un dispositivo NFC como para una etiqueta NFC, por lo que la información de NDEF es independiente del tipo de dispositivos que se estén comunicando.

Dentro del formato de un mensaje NDEF se puede enviar información de distintos tipos, desde encapsular documentos XML[5], datos encriptados, imágenes o cadenas de información. [5]

RTD (*Record Type Definition)* especifica los tipos de registros que pueden ser enviados en los mensajes entre los dispositivos. Encontramos:

- *Smart Poster RTD*: Para incorporar etiquetas con datos (URLs, SMS o números de teléfono).
- *Text RTD*: Para registros que solo contienen texto.
- *Uniform Resourc e Identifier (URI) RTD*: Para registros referido a un recurso de internet.

2.1.3. CARACTERÍSTICAS DE FUNCIONAMIENTO

Como NFC se basa en las tecnologías sin contacto, se requieren dos tipos de dispositivos para el establecimiento de la comunicación. El dispositivo que inicia la comunicación es el encargado de monitorizar la misma y éste rol es intercambiable entre las dos partes implicadas. Partimos de dos elementos:

- **El iniciador** (*Initiator*) como su nombre indica es el que inicia y controla el intercambio de información.
- **El objetivo** (*target*) es el dispositivo que responde a la petición del iniciador.

Cualquier dispositivo provisto de la tecnología NFC puede operar de las dos formas.

2.1.3.1. MODOS DE FUNCIONAMIENTO

Como en la tecnología RFID, existen dos modos de operar:

En el **modo activo**, cada uno de los dispositivos genera su propio campo de radiofrecuencia (RF) para enviar los datos (*peer to peer*), reconociéndose automáticamente.

[5] XML (Extensible Markup Languaje): Metalenguaje de etiquetas para documentos de formato electrónico. *http://www.w3c.es/divulgacion/guiasbreves/tecnologiasxml*

Un dispositivo desactiva su campo de RF, mientras está esperando una respuesta. En este modo, ambos dispositivos necesitan tener una fuente de energía para su funcionamiento.

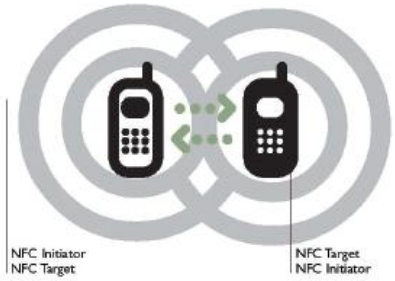


Figura 2.3: Modo Activo NFC [6]

En el **modo pasivo** solamente uno de los dispositivos genera el campo magnético. El que inicia la conexión será el encargado de generar ese campo permitiendo que el dispositivo destino aproveche la modulación de carga para transferir datos. Este modo de funcionamiento es muy importante para dispositivos de ahorro de energía, como los teléfonos móviles y PDAs, donde se hace necesario priorizar el uso de la energía.

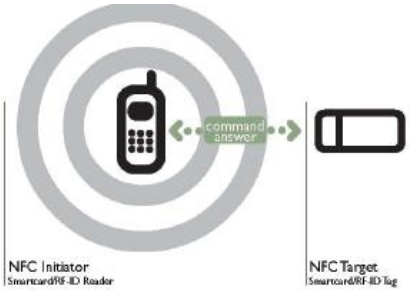


Figura 2.4: Modo Pasivo NFC [6]

2.1.3.2. PROTOCOLOS

En la estandarización de la comunicación NFC se han definido dos protocolos, NFCIP1 (*Near Field Communication Interface and Protocol-1*) y NFCIP2 (*Near Field Communication and Protocol-2*).

NFCIP-1: Este protocolo define tanto el enlace de RF con la que NFC trabaja, en este caso con 13, 56 MHz, como la arquitectura para las tasas de transferencia de datos de 106, 212 y 424 Kbit/s, y los modos de operación activo y pasivo. En los sistemas NFC se pueden comunicar como máximo dos dispositivos simultáneamente.

Uno de los pares en la comunicación se llama *Iniciator* y tiene un rol activo, mientras que el par pasivo se llama *Target*. Estos roles son asignados, incluso cuando los dos dispositivos NFC que se comunican están provistos de carga de baterías.

NFCIP-2: Este estándar especifica un método para escoger uno de los tres posibles modos de comunicación definidos (*reader, tag, NFC-chip*), para que no interfieran otras comunicaciones en curso en la frecuencia 13,56 MHz.

2.1.3.3. PASOS DE LA COMUNICACIÓN

La comunicación se realiza en 5 fases:

Descubrimiento: Los dispositivos inician la etapa de rastreo mutuo para su reconocimiento posterior.

Autenticación: Los dispositivos verifican si deben establecer algún tipo de cifrado para la comunicación con el otro dispositivo o están autorizados.

Negociación: Se definen los parámetros como la velocidad de transmisión, la identificación del dispositivo, el tipo de aplicación, su tamaño, y si es el caso también definen la acción a ser solicitada.

Transferencia: Se realiza el intercambio de datos.

Confirmación: El dispositivo receptor confirma el establecimiento de la comunicación y la transferencia de datos.

2.1.3.4. SEGURIDAD NFC

Podemos considerar la tecnología NFC segura, ya que por su corto alcance los dispositivos tienen prácticamente que tocarse. Aunque por sí sola no asegura comunicaciones seguras.

No ofrece protección contra los que se dedican a escuchar comunicaciones y es también vulnerable a modificación de datos. Las aplicaciones deben usar protocolos criptográficos de una capa superior para establecer un canal seguro, como puede ser la SSL.

Pero esto se contrarresta con la distancia de operación de NFC ya que al ser de tan solo unos pocos centímetros el espía debería estar dentro de ese rango. Además un dispositivo pasivo, que no genera su propio campo de radio frecuencia, es mucho más difícil de intervenir que un dispositivo activo.

Podemos afirmar por todas estas características que NFC es una tecnología intuitiva. “Si quieres un servicio, solo tienes que tocarlo”. Y muy sencilla porque no precisa configuración por parte del usuario.

2.1.4. ACTUALIDAD TECNOLOGICA DE NFC

2.1.4.1. MODELOS DE SERVICIOS

En los dispositivos móviles NFC, el elemento seguro es uno de los componentes más importantes. Actualmente, existen tres opciones para su ubicación dentro del teléfono:

- **Elemento seguro incorporado en la electrónica del móvil**: Esta opción ha sido utilizada por la gran mayoría de proyectos pilotos que actualmente se desarrollan en todo el mundo, ya que de esta forma el elemento seguro tiene todas las certificaciones necesarias hardware y software demandadas por el sector bancario. Sin embargo si se quiere cambiar de dispositivo móvil, se pierden todas estas credenciales.
- **Tarjeta de memoria como elemento seguro**: Una tarjeta de memoria (SD, MMC o MS) incorpora un chip seguro con un micro controlador de memoria flash. Esta solución permite que terceras partes puedan suministrar tarjetas precargadas con su aplicación.
- **Tarjeta SIM como elemento seguro**: Con esta arquitectura, la tarjeta SIM incorpora la aplicación de pago, así como cualquier otra aplicación NFC. La aplicación puede almacenarse en el propio SIM o como un componente adicional en el conector del SIM.

Como vemos hoy en día todavía existen algunas deficiencias y problemas que pueden ser derivados de las barreras técnicas y tecnologías, ya que las aplicaciones son especificas para ciertos dispositivos, de acuerdo a un modelo o marca. Esto supone un freno para los proveedores de servicios al desarrollar y probar una aplicación, ya que debe ser desarrollada de forma única para cada dispositivo.

Se presentan por lo tanto varios desafíos que es necesario afrontar para la adopción masiva de esta tecnología. Es necesario avanzar en la estandarización de la arquitectura y ubicación del elemento seguro para facilitar el desarrollo de aplicaciones. Para esto el *NFC Forum* jugará un papel fundamental en la estandarización de la comunicación entre el chip NFC y el procesador base del teléfono, HCI (*Host Controller Interface*). Además la GSMA[6] deberá definir y estandarizar la comunicación entre la tarjeta SIM y el chip NFC a través del protocolo denominado *Single Wire Protocol* (SWP) con el fin de conseguir una integración plena del Elemento Seguro dentro de la arquitectura de seguridad.

2.1.4.2. ACTUALIDAD DE LA INDUSTRIA

En la figura siguiente encontramos los diferentes actores que participan en la industria NFC.

Figura 2.5: Ecosistema NFC

Entre los diferentes actores, podemos destacar:

Cliente Final: Quien utiliza el dispositivo móvil para las comunicaciones y utiliza los servicios móviles de NFC, subscribiéndose a un MNO. La recomendación de la GSMA es que la SIM actúe como elemento seguro.

MNO (*Mobile Network Operator*): Facilita el despliegue de las aplicaciones sobre la SIM, tiene el control sobre el espacio de las *SIM Cards*, utilizadas para almacenar las aplicaciones de seguridad. Controlan las descargas y almacenamiento de servicios y aplicaciones.

Proveedor de Servicio: Ofrece los servicios NFC al cliente, como las empresas de tarjetas (*Card Companies*).

[6] *http://www.gsmworld.com/*

TSM (*Trusted Service Manager*): Provee, distribuye y gestiona de forma segura los servicios prestados por el Proveedor de Servicio a la base de clientes del MNO. Idealmente el Proveedor de servicio interactúa con un único TSM. Es importante ya que proporciona un único punto de contacto de los Proveedores de Servicios con la base de datos de clientes de los MNOs. Además gestiona y administra las descargas de las aplicaciones, y asegura que estas se den de forma segura, gestionando también el ciclo de vida de las mismas.

Handset, NFC Chipset and UICC Manufacturer: Produce los dispositivos NFC y el hardware UICC (o SIM Card) asociado. EL UICC provee tanto seguridad a nivel lógico como a nivel físico, ya que las aplicaciones NFC necesitan trabajar bajo un entorno seguro (*Secure Environment*)

Desarrollador de la aplicación: Es el encargado de los diseños y desarrollo de las aplicaciones móviles NFC.

Organismos nacionales de normalización: Desarrolla un estándar global para NFC, que permite la interoperabilidad, compatibilidad y futuros desarrollos de aplicaciones y servicios NFC. [7]

Con todo esto, afirmamos que el papel que jugará el TSM dentro de la cadena de valor se debe definir y acordar entre los principales actores (proveedores de servicio y MNOs) a fin de prestar los servicios NFC de forma eficiente a los clientes, para crear un modelo de negocio consistente. [8]

2.1.4.3. ESTADO ACTUAL DE NFC EN EL MUNDO

Existen a nivel mundial proyectos interesantes sobre la tecnología NFC, entre los que podemos destacar:

- Emt de Malaga, en mayo del 2008 puso en marcha el proyecto “Recarga, consulta y validación en autobuses urbanos”, en la que participaron las empresas Orange, Indra, Mobipay y Oberthur Technologies. [9] [10]
- Hotel Clarion Estocolmo, a mediados del 2010, lanza un proyecto piloto para ofrecer a sus huéspedes la posibilidad de comprar y obtener la llave de la habitación en sus dispositivos móviles antes de llegar al hotel. En este proyecto colaboran las empresas: Assa Abloy, TeliaSonera, Choice Hotels Scandinavia, Venyon y VingCard Elsafe. [11]

- Telefónica lanza en Barcelona del 2010 su primera experiencia de Pago con móvil NFC, basada en los estándares impulsados por la industria. En este proyecto colaboran empresas como Samsung, La Caixa, Visa, Sermepa, Ingenico, Giesecke & Devrient y Gemalto.
- Emt Madrid, en marzo 2011, anuncia el proyecto piloto para pagar el autobús urbano a través del móvil, en la que colaboran empresas como Ericsson y Bankinter. [12]
- Google a mediados del 2011, anuncia el servicio de pagos sin contacto "Google Wallet", en colaboración con First Data, Citibank y Mastercard, junto con la operadora Sprint en Estados Unidos. Totalmente compatible con el servicio de pago *PayPass* que Mastercard ofrece. [13]
- La banca europea se une a los pagos por NFC, El *European Payments Council* (EPC), que representa la industria de la banca europea, ha publicado un borrador de directrices que espera que sirva para potenciar el uso de la tecnología NFC como sistema de pago. Se espera que la versión final del documento esté publicada a finales de octubre. [14]

Además las grandes compañías de teléfonos móviles están incorporando en sus dispositivos esta tecnología, para poder lanzar proyectos como es el caso de Google descrito arriba.

Samsung está preparando para este año el lanzamiento de tres teléfonos Bada[7] que proporcionen la tecnología NFC. [15]

2.2. BLUETOOTH

2.2.1. CARACTERÍSTICAS Y ESPECIFICACIONES

Bluetooth forma parte de las tecnologías creadas para la comunicación inalámbrica de uso personal. Es una tecnología desarrollada por Ericsson en 1994, que facilita el intercambio de información entre algunos dispositivos como computadores, teléfonos móviles o PDAs. Se creó con el objetivo de sustituir los cables que conectan los dispositivos manteniendo al mismo tiempo altos niveles de seguridad.

[7] Bada: Sistema operativo propio de Samsung

Con todo ello, en el año 1999 se creó el SIG de Bluetooth (*Special Interest Group*), que consistía, en la unión de diversas empresas (entre ellas, Ericsson, Nokia, Toshiba e IBM). [16]

Forma parte de las llamadas WPAN (*Wireless Personal Area Network*), con el objetivo de lograr la interoperabilidad con otros dispositivos inalámbricos. Orientada tanto a la comunicación de datos como de voz, permite establecer pequeñas redes inalámbricas entre equipos personales, permitiendo la sincronización entre ellos. Por lo tanto facilita las comunicaciones entre equipos fijos y móviles.

Permite comunicaciones, incluso a través de obstáculos, a distancias de unos 10 metros. Es un estándar que fue designado para un consumo bajo de potencia. La especificación de Bluetooth define una estructura uniforme para la comunicación de una amplia gama de dispositivos[8].

Opera en la banda libre de radio ISM[9] a 2,4 Ghz, a una velocidad máxima de transmisión que depende de la versión Bluetooth que utilicemos (versión 1.2: 1 Mbps; versión 2.0: hasta 3 Mbps; versión 3.0: 30Mbps). Al ser la banda ISM libre, está abierta a cualquier usuario que la quiera utilizar y expuesta al ruido externo pudiendo ocasionar interferencias y pérdidas de información. Por ello la tecnología Bluetooth realiza un rápido emparejamiento y utiliza saltos de frecuencia en la transmisión para garantizar una conexión robusta.

Cada dispositivo Bluetooth tiene asignada una dirección Bluetooth única de 48 bits, basada en el estándar IEEE 802.11 para WLAN. Esta dirección se utiliza tanto para la identificación, como para sincronizar el salto de frecuencia entre los dispositivos y la generación de claves en los procedimientos de seguridad.

2.2.2. ARQUITECTURA

Para garantizar la interoperabilidad de unos fabricantes con otros, y que los dispositivos Bluetooth puedan descubrirse unos a otros, explorar servicios y hacer uso de ellos, Bluetooth esboza un sistema radio, una pila de protocolos y varios perfiles.

[8] *http://www.bluetooth.com/Pages/Product-Directory.aspx*
[9] Banda internacional médico-científica

2.2.2.1. NIVEL DE HARDWARE

Se distinguen dos partes:

- Dispositivo radio: su función es modular y transmitir la señal.
- Controlador digital: entre sus tareas destacan el envío y recepción de datos, determinación de conexiones, autenticación, negociación y determinación de tipos de enlace o del tipo de cuerpo de cada paquete.

2.2.2.2. PROTOCOLOS

La especificación Bluetooth utiliza una arquitectura de protocolos que divide las diversas funciones de red en un sistema de niveles. Los dispositivos que vayan a participar en las comunicaciones tendrán que ejecutar la misma pila de protocolos.

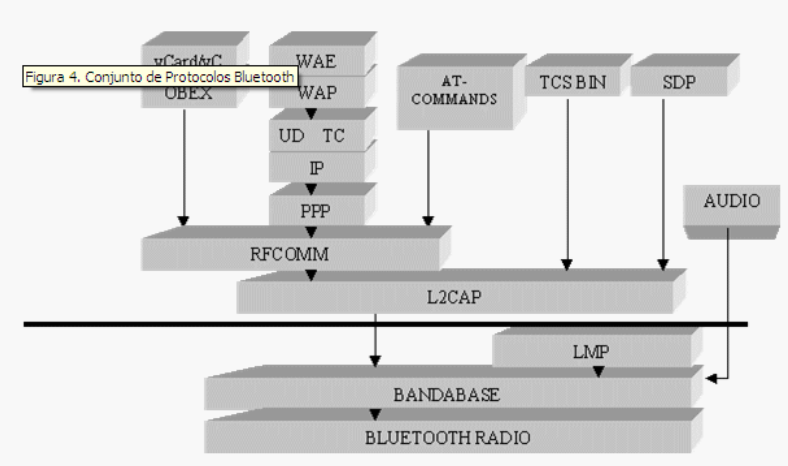


Figura 2.6: Pila de Protocolos Bluetooth [17]

El Grupo Bluetooth SIG, ha desarrollado los protocolos de la primera capa, los cuales son usados por la mayoría de los dispositivos Bluetooth. Mientras que las capas de reemplazo de Cable, Control de Telefonía, y de Protocolos adaptados conforma los llamados protocolos orientados a la aplicación. Dichos protocolos son abiertos, y permiten la inclusión de otros nuevos, por lo que el estándar es muy flexible.

2.2.2.3. PERFILES

Además de la pila de protocolos, existen una serie de perfiles que definen los diferentes tipos de uso y aplicaciones de la tecnología inalámbrica Bluetooth. Así se pueden crear aplicaciones compatibles con otros dispositivos que se ajusten a este estándar.

Hay un gran número de perfiles, pero hay cuatro básicos:

- GAP (*Generic Acces Profile*): Perfil de acceso genérico.
- SPP (*Serial Access Profile*): Perfil de puerto de serie.
- SDAP (*Service Discovery Application Profile*): Perfil de aplicación de descubrimiento de servicio.
- GOEP (*Generic Object Exchange Profile*): Perfil genérico de intercambio de objetos.

GAP es la base para los demás perfiles, todos están basados en este. Define procedimientos generales para establecer conexiones entre dos terminales, incluyendo el descubrimiento de dispositivos Bluetooth, gestión y configuración, y los procedimientos relacionados con los diferentes niveles de seguridad.

SDAP describe operaciones fundamentales necesarias para el descubrimiento de servicios. Así como los protocolos y los procedimientos que usaran las aplicaciones para localizar los servicios en otros dispositivos Bluetooth.

SPP define los requerimientos de los dispositivos Bluetooth necesarios para configurar y emular conexiones RFCOMM entre dos dispositivos. Reemplaza el cable de la conexión RFCOMM por la tecnología inalámbrica Bluetooth.

GOEP es un perfil abstracto que se utiliza para transferir objetos de un dispositivo a otro. Este perfil define todos los elementos necesarios para soportar OBEX.

2.2.3. SEGURIDAD BLUETOOTH

Es importante que las transmisiones se realicen de forma segura, por ello Bluetooth dispone de diferentes métodos de seguridad a la hora de poner en práctica mecanismos de seguridad. Existen tres modos de seguridad para las conexiones:

No Seguro: Los mecanismos de seguridad están deshabilitados.

Seguridad a nivel de enlace: La seguridad se inicia antes de establecerse en cualquier canal. Además del cifrado hay autenticación con PIN y seguridad MAC.

En este modo de seguridad podemos destacar tres mecanismos:

- **Autenticación**: Se emplea una clave secreta por ambos dispositivos al establecerse la primera comunicación.
- **Autorización**: Se establecen niveles de confianza que determinan la capacidad de acceso de un dispositivo a los servicios. (confianza total, confianza parcial o confianza nula).
- **Cifrado de datos:** Garantiza la confidencialidad del mensaje transmitido, protegiendo la información que transmite.

Seguridad a nivel de servicio (L2CAP): No hay codificación adicional de PIN o claves. La seguridad se inicia después de establecerse el canal.

Es el propio fabricante de cada producto quien determina el modo de seguridad del mismo. Tanto los dispositivos como los servicios que ofrece, cuentan con distintos niveles de seguridad. [18]

2.2.4. APLICACIONES CON JAVA

Actualmente existe una amplia gama de productos comerciales que abarcan aéreas como audio y video, dispositivos periféricos, dispositivos médicos, equipo de oficina y computo, dispositivos y accesorios portátiles de comunicación, aparatos de medición y juegos, entre otros. Por si mismo ofrece aplicaciones nativas como la transferencia de archivos, la comunicación de voz o la conectividad de equipos periféricos entre otras.

Sin embargo, la integración de Bluetooth con otras herramientas, como J2PME (*Java 2 Platform Micro Edition*) abre una ventana inmensa de posibilidades para la creación de aplicaciones que puedan ser hechas a la medida de los usuarios.

El desarrollo de aplicaciones bajo la especificación de Java JSR 82 con J2ME[10] permite el acceso y control sobre dispositivos que soporten Bluetooth, además de la configuración J2ME conocida como *Connected Limited Device Configuration* (CLDC)[11].

[10] J2ME: *http://java.com/es/download/faq/whatis_j2me.xml*

Las interfaces generadas para las aplicaciones a través de Java (*APIs Application Programming Interfaces*) conocidas como *Midlets*[12], posibilitan el registro y descubrimiento de servicios, descubrimiento de dispositivos, establecimiento de canales de comunicación, además del envío y recepción de datos (no incluyen voz).

Las aplicaciones de Java para Bluetooth utilizan dos paquetes esenciales:

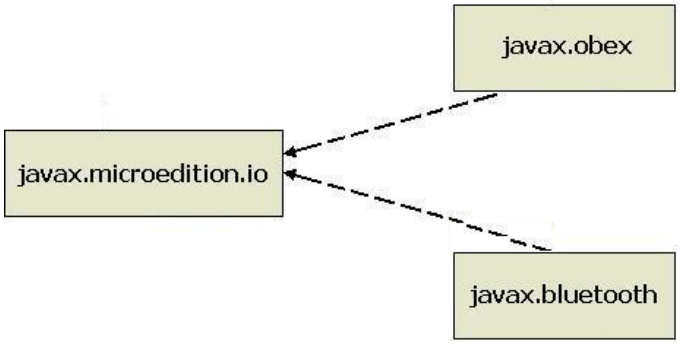


Figura 2.7: Paquetes API Bluetooth

- **javax.bluetooth**: Conforma las especificaciones básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación. Esta comunicación es a bajo nivel, mediante flujos de datos o mediante la transmisión de cadenas de bytes.
- **javax.obex:** Utilizado para el intercambio de objetos (transferencia de datos) entre dispositivos. Es muy similar a HTTP.

Para la programación de una API[13], se deben contemplar cinco funciones principales, basadas en la operación del protocolo Bluetooth:

1. Inicializar los parámetros de comunicación como son la velocidad de transmisión, el puerto de comunicación y el establecimiento del modo de descubrimiento de dispositivos.
2. Establecer la definición de los dispositivos para etiquetarlos como locales o remotos.
3. Descubrimiento de los dispositivos en la red.
4. Registro y descubrimiento de los servicios disponibles en la red.
5. Comunicación.

[11] CLDC: *http://java.sun.com/products/cldc/*
[12] Midlet: programa en lenguaje de programación Java para dispositivos embebidos.
[13] API (*Application Programming Interface*), conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

2.2.5. VERSIONES DE LA TECNOLOGIA BLUETOOTH

Desde 1994, cuando vio la luz por primera vez, hasta la actualidad se han ido implementando diferentes versiones de la tecnología Bluetooth. Cada una de ellas con notables mejoras en lo que respecta al consumo de energía y velocidades de transferencia, así como algunas mejoras de seguridad.

Bluetooth versión 1.1: Ericsson comienza un estudio para investigar la viabilidad de una nueva interfaz de bajo coste y consumo para la interconexión vía radio entre dispositivos como teléfonos móviles y otros accesorios. Se llegó a un enlace de radio de corto alcance, llamado MC link. Este enlace se basa en un chip radio, por lo que se fue viendo que se podía utilizar ampliamente en un gran número de aplicaciones.

Bluetooth versión 1.2: Provee una solución inalámbrica complementaria para coexistir con la tecnología Wifi en el espectro de los 2,4 GHz, sin interferencias entre ellos. Ofrece una calidad de voz (*Voice Quality- Enhanced Voice Processing*) con menor ruido ambiente, y provee una configuración más rápida de la comunicación con los otros dispositivos Bluetooth dentro del rango de alcance, como pueden ser PDAs, impresoras o dispositivos móviles. Utiliza la técnica "*Adaptative Frecuency Hopping*" (AFH)[14], que ejecuta una transmisión más eficiente y un cifrado más seguro.

Bluetooth versión 2.0: Esta versión incorpora la técnica "*Enhanced Data Rate*" (EDR) que le permite mejorar las velocidades de transmisión en hasta 3 MBps.

Bluetooth versión 2.1: Simplifica los pasos para crear la conexión entre dispositivos, además el consumo potencial es 5 veces menor.

Bluetooth versión 3.0: Aumenta considerablemente la velocidad de transferencia. La idea es que el nuevo Bluetooth trabaje con Wifi, de tal manera que sea posible lograr mayor velocidad en los *smartphones*, pudiendo ser la transferencia de datos de hasta unos 24 Mbps.

Bluetooth versión 4.0: Disminuye el consumo de energía, ampliando la cantidad de aplicaciones, permitiendo la incorporación de receptores y transmisores Bluetooth en dispositivos pequeños como relojes o reproductores portátiles. Se mejora la seguridad con la encriptación AES-128[15].

[14] AFH, salto de frecuencia adaptable, reduce los efectos de interferencia entre Bluetooth y otros tipos de dispositivos.

[15] AES: esquema de cifrado por bloques, es uno de los algoritmos más populares usados en criptografía simétrica.

2.3. TECNOLOGÍA JAVA

La tecnología Java constituye una plataforma informática presentada por Sun Microsystems en 1995. Esta tecnología está compuesta por dos partes:

- El lenguaje de programación Java.
- La plataforma Java.

2.3.1. CARACTERISTICAS

Sin entrar en demasiados detalles sobre el lenguaje Java, sus principales características son:

Familiar: se asemeja a la sintaxis de C y C++, pero su modelo de objetos es más simple y elimina herramientas de bajo nivel, como la manipulación directa de punteros o memoria.

Orientado a objetos: Se refiere al un método de programación y al diseño del lenguaje. Los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Que puede verse como un paquete que contiene el “comportamiento” (código) y el “estado” (datos).

Robusto: Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución.

Portable: Puede ejecutarse sobre diferentes sistemas operativos, ocultando la complejidad del dispositivo de las aplicaciones y desarrolladores de aplicaciones.

Distribuido: Proporciona una colección de clases para su uso en aplicaciones de red, que permite abrir *sockets* y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Alto rendimiento: Soporta sincronización de múltiples hilos de ejecución a nivel de lenguaje. Además puede ser usado para crear dos tipos de programas: aplicaciones independientes y applets. [19]

2.3.1. PLATAFORMA JAVA

La plataforma Java constituye el ambiente hardware y software en el cual se ejecutan las aplicaciones. Tiene dos componentes la máquina virtual (JVM) y el Java API (*Application Programming Interfaces*).

El JVM es la base de la plataforma Java y es necesario que el destino de ejecución de las aplicaciones Java tenga implementado la JVM para que puedan ser ejecutadas. El Java API, por su parte, es una gran colección de librerías, conocidas como paquetes.

Desde el nacimiento de la plataforma Java, se ha ido adaptando a las necesidades tanto de los usuarios como de las empresas ofreciendo soluciones y servicios por igual. Por ello se han desarrollado soluciones personalizadas para cada ámbito tecnológico. Estas ediciones son:

- Java SE (*Java Standard Edition*): Es la edición que se emplea en ordenadores personales. Es la versión que debes instalar para programar en Java aunque utilices alguna de las otras ediciones.
- Java ME (*Java Micro Edition*): Es la edición que se emplea en dispositivos limitados de recursos, como teléfonos móviles, PDAs o electrodomésticos.
- Java EE (*Java Entreprise Edition*): Orientado a entornos distribuidos empresariales o de Internet.

A la hora de desarrollar aplicaciones Java, Sun ofrece el SDK de Java, o JDK (*Java Development Kit*) en cuyo seno reside el **JRE**, que es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java, e incluye herramientas como el compilador de Java, Javadoc, o el depurador.

2.4. J2ME

J2ME es la edición de Java especialmente diseñada y adaptada para aprovechar al máximo los escasos recursos con los que cuentan los dispositivos integrados. Representa una versión simplificada de J2SE.

La diferencia de las otras versiones en el uso de componentes básicos como una maquina virtual denominada KVM(*Kilo Virtual Machine*) en vez del uso de la JVM (*Java Virtual Machine*).

Incluye interfaces de usuario flexibles, seguridad robusta, una función de protocolos de red y soporte para aplicaciones de red y fuera de línea que se pueden descargar de forma dinámica.

2.4.1. ARQUITECTURA

El entorno de ejecución de J2ME se compone de una selección de máquina virtual, configuración, perfil y paquetes opcionales.

A continuación se detallan cada uno de los componentes de la arquitectura:

1. Maquina Virtual: interpreta el código intermedio (*bytecode*) de los programas Java precompilados a código máquina ejecutable por la plataforma, efectúa las llamadas pertinentes al sistema operativo y observa las reglas de seguridad y corrección de código definidas para el lenguaje Java. Encontramos la KMV (*Kilo Virtual Machine*, maquina virtual de la configuración CLDC) y CVM (*Compact Virtual Machine*, maquina virtual de la configuración CDC).

2. Configuración: Conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos. Estas APIs describen las características básicas, comunes a todos los dispositivos. Encontramos la CLDC (*Connected Limited Device Configuration*, orientado a dispositivos pequeños como móviles o PDAs) y la CDC (*Connected Device Configuration*, orientado a dispositivos de mayor capacidad).

3. Perfiles: Define las APIs que controlan el ciclo de vida de la aplicación o interfaz de usuario. El perfil establece unas APIs que definen las características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Existen varios perfiles para cada configuración.

Este entorno de ejecución se estructura en capas, una construida sobre la otra como vemos en la figura 2.8.

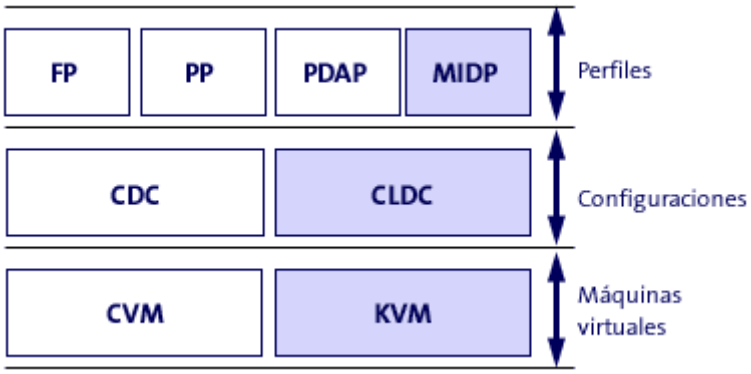


Figura 2.8: Arquitectura de J2ME

En los dispositivos de este proyecto se utiliza la configuración CLDC, por lo que hace necesario el uso de la KVM. El perfil MIDP (*Mobile Information Device Profile*) está construido sobre la configuración CLDC. Y las aplicaciones que se realizan utilizando este perfil reciben el nombre de *MIDlets.*

De aquí en adelante se utilizara el concepto de *MIDlet* para referirse a las aplicaciones móviles que estén implementadas bajo alguna de las versiones del perfil MIDP. [20]

2.5. TECNOLOGIAS EN EL LADO DEL SERVIDOR

Se han utilizado distintas tecnologías para el desarrollo de la aplicación web, que se detallan a continuación.

Una aplicación web es un conjunto de *Servlets*, *JSPs* y otros recursos (como ficheros HTML, imágenes o ficheros de configuración) relacionados entre sí.

Todo ello se puede ejecutar en un servidor web que ofrece una determinada funcionalidad a la que los clientes acceden típicamente a través de un navegador. Este servicio se ha desarrollado con Apache Tomcat.

Se ha desarrollado también una base de datos, mediante MySQL.

2.5.1. APLICACIÓN WEB

2.5.1.1. HTML

HTML (*HyperText Markup Language)* es el lenguaje base para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto cono objetos tales como imágenes. Además también puede describir, hasta un cierto punto, la apariencia del documento.

El diseño en HTML debe cumplir unas especificaciones propias del lenguaje y respetar unos criterios de accesibilidad web, siguiendo unas pautas vigentes. Se encuentra disponible y desarrollado por el W3C[16]. HTML se escribe en forma de "etiquetas".

La estructura básica de una página web comienza con la etiqueta *<html>* y acaba con *</html>*. Dentro de estas dos etiquetas ira la cabecera de la pagina, *<head></head>*, donde se pondrán el titulo de la pagina, los metadatos o estilo y el cuerpo de la pagina, *<body></body>*, donde ira el resto de contenidos de la pagina.

2.5.1.2. SERVLETS

Los *Servlets* son programas que se ejecutan en un servidor Web o un Contenedor JEE(*Java Entreprise Edition*), diseñados para ofrecer contenido dinámico desde un servidor web, generalmente HTML. Como están implementados con la tecnología Java, pueden ser ejecutados en cualquier servidor. Entre sus características más destacables encontramos que son muy eficientes, incrementan la funcionalidad de una aplicación web, se cargan de forma dinámica por el entorno de ejecución Java del Servidor cuando se necesitan. Cada petición es manejada por un *thread* Java de peso ligero. Cuando se recibe una petición del cliente, el contenedor inicia el *Servlet* requerido. Este procesa la petición del cliente y envía la respuesta de vuelta al contenedor, que es enviada al cliente.

[16] http://validator.w3.org/

Los *Servlet Java* utilizan mensajes *request* y *response* del protocolo HTTP, mediante los interfaces *HttpServletRequest* y *HttpServletResponse.* Estos mensajes se suceden síncronamente (no hay un response sin un *request* previo y no puede haber más de un *request* consecutivo sin su response previo y viceversa).

Su uso en aplicaciones web implica una mejora sustancial en la gestión de recursos, ya que no se genera un único proceso en el servidor cada vez que llega una petición del cliente, sino que se crea un hilo ligero. Como es un hilo controlado por el servidor el cambio de contexto entre procesos se hace muy fácil.

Por todos estos motivos, la independencia de la plataforma utilizada, el alto soporte de excepciones y el tipo de funcionamiento se utilizo esta tecnología para implementar la aplicación web.

2.5.1.3. JSPs

Mientras que los *JSP* (*Java Server Pages*) es una tecnología que nos permite mezclar HTML estático con HTML generado dinámicamente. La posibilidad de usar APIs de Java hace de *JSP* una poderosa herramienta de desarrollo ya que se obtiene la ventaja de la programación orientada al objeto, como creación de clases especiales llamadas componentes o *Java Beans*.

Los *Java Beans* son clases que contienen propiedades (normalmente atributos de la instancia privados), el acceso a las propiedades se realiza mediante métodos de acceso *get* y *set*, y siempre tiene un constructor sin argumentos (aunque puede tener más).

Los *JSPs* pueden ser vistas como una abstracción de alto nivel de los *Servlets*, ya que despues de compilada, un *JSP* es convertida en un *Servlet*, con todas las ventajas que implica, por este motivo se usó para el presente proyecto. [21]

2.5.1.4. ARQUITECTURA DE UNA APLICACIÓN CON SERVLETS Y JSP

Una aplicación Web puede combinar *Servlets* y páginas *JSP*, de la siguiente forma:

- Procesado de parámetros de la petición: *Servlets.*
- Acceso a base de datos: *Servlets.*

- Lógica de la aplicación: *Servlets*.
- Presentación (vistas): *JSP.*

El modelo de funcionamiento es el siguiente:

1. El cliente envía la petición HTTP a un *servlet*.
2. El *servlet* procesa la petición, si es necesario, se conecta a la base de datos.
3. El *servlet* redirige la petición a un JSP. Si es necesario, añade *beans* como parámetros.
4. El *JSP* lee los parámetros y devuelve la respuesta formateada visualmente al usuario. [22]

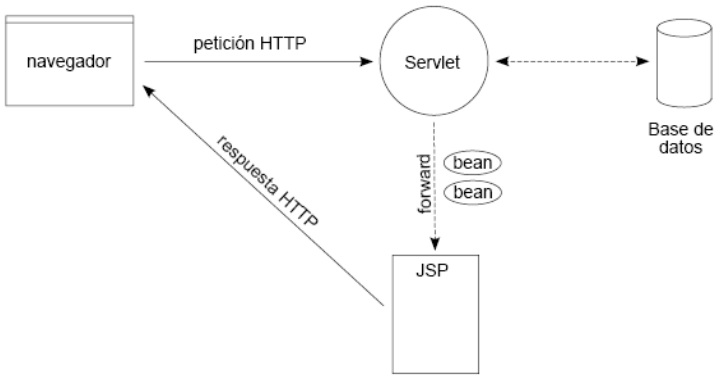


Figura 2.9: Arquitectura de una aplicación con *Servlets* y *JSPs*

Todas estas ventajas que nos ofrecen tanto los *Servlets Java*, como las páginas *JSP*, llevan a decidir el uso de esta tecnología en el presente proyecto, debido al alto rendimiento de la tecnología en entornos web como la cantidad de documentación y recursos sobre ambas tecnologías.

Para ejecutar y ver cómo funcionan nuestras pruebas, necesitaremos un servidor web con soporte para *JSP* y *Servlets*. Por lo que instalamos en nuestro ordenador nuestro propio servidor, Apache Tomcat como elemento contenedor de la aplicación a implementar.

2.5.2. TOMCAT

Tomcat (también llamado *Jakarta Tomcat* o *Apache Tomcat*) funciona como un contenedor de *Servlet* desarrollado bajo el proyecto *Jakarta* en la *Apache Software Foundation*[17]. *Tomcat* implementa las especificaciones de los *Servlets* y de *JSP* de *Sun Microsystems*.

Dado que *Tomcat* fue escrito en Java, funciona en cualquier sistema operativo que disponga de la maquina virtual Java.

Es un servidor web con soporte de *Servlets* y *JSPs*. El motor de *Servlets* de *Tomcat* a menudo se presenta en combinación con el servidor web Apache[18]. Puede funcionar como servidor web por sí mismo, en entornos con alto nivel de tráfico y alta disponibilidad.

La aplicación incluye varias herramientas para gestionar y configurar el entorno, pero también lo permite editando sus ficheros de configuración en xml.

Destacamos entre las características más importantes, la posibilidad de funcionar en distintos entornos de desarrollo, además cuenta con *Apache Software Foundation* soportando su desarrollo lo que proporciona una garantía de calidad y estabilidad. Su licencia es de código abierto, lo que permite el uso y distribución del código fuente tanto para software libre como para software propietario.

Además viene con la parte de J2EE necesaria para trabajar con *Servlets* y *JSP*, por ello se escogió *Tomcat* como contenedor de la aplicación. [23]

2.5.3. MYSQL

MySQL es un gestor de base de datos relacional, multi-hilo y multiusuario, sencillo de usar. MySQL AB es la empresa que desarrolla MySQL como software libre en un esquema de licencia dual, por un lado ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero si las empresas desean incorporarlo en productos privados pueden comprar a la empresa una licencia específica que les permita este uso.

[17] http://www.apache.org/
[18] http://tomcat.apache.org/

MySQL es muy utilizado en aplicaciones web. Las características principales son:

Es un gestor de base de datos, capaz de manejar un conjunto de datos de manera eficiente y cómoda.

Es una base de datos relacional, los datos están almacenados en tablas entre las cuales se establecen unas relaciones para manejar los datos de una forma eficiente y segura. Para usar y gestionar una base de datos relacional se usa el lenguaje estándar de programación SQL.

Es Open Source, código fuente de MySQL se puede descargar y esta accesible a cualquiera, por otra parte, usa la licencia GPL para aplicaciones no comerciales.

Adaptación a entornos de desarrollo, permite su interacción con los lenguajes de programación más utilizados como PHP, Perl y Java y su integración en distintos sistemas operativos.

Cumple con las características esenciales que deben exigirse a un sistema gestor de base de datos, como soporte multilenguaje, soporte multiplataforma, escalabilidad, estabilidad y soporte transaccional. Además de la licencia con la que se distribuye. [24]

2.6. TELEFONO MÓVIL NOKIA 6131 NFC

El proyecto está orientado a la aplicación de la tecnología NFC mediante el uso de teléfonos móviles que sean compatibles, por lo tanto el dispositivo de la comunicación de nuestra aplicación es un teléfono móvil de la marca Nokia serie 6131, el cual viene incorporado con el modulo y la antena NFC internamente.

En el mercado existen algunas marcas de teléfonos móviles en los que viene incorporada la tecnología NFC. Para la elección del modelo del teléfono NFC con el cual trabajar y realizar la implementación del sistema se tomaron en cuenta algunos aspectos como facilidad de adquisición, aplicaciones reales en las que hayan sido probados los dispositivos o compatibilidad con otros equipos.

Nokia es uno de los promotores de esta tecnología, formando parte del foro NFC, que fue abierto sin fines de lucro para el desarrollo de esta tecnología, dando un espacio amplio para el contacto con personas de distintos campos como la programación.

Además ha creado herramientas que permiten el manejo y el desarrollo de aplicaciones para cada uno de sus terminales móviles como el kit de desarrollo de software para el Nokia 6131, el SDK 6131 entre otras. Por este motivo se escogió esta plataforma.

Lo más importante a destacar de este dispositivo es:

- Soporta el API JSR 257 para aplicaciones que utilicen NFC.
- Soporta lectura y escritura en los formatos mas populares de etiquetas: *Mifare* 1K, *Mifare* 4K, *Ultralight*, *Topaz*, *Jewel* y *Felica.*
- Permite aplicaciones peer to peer.
- Contiene un chip seguro integrado que permite ser tratado como una tarjeta ISO 14443 Tipo A o *Mifare* 4K.

2.6.1. NOKIA SDK NFC

El Kit Nokia 6131 NFC SDK ofrece un entorno de desarrollo total, tanto la simulación con etiquetas NFC como con tarjetas NFC inteligentes, así como la comunicación con lectores de tarjetas externas y dispositivos que soportan JSR-257[19]. Tal y como se muestra en la figura 3.1.

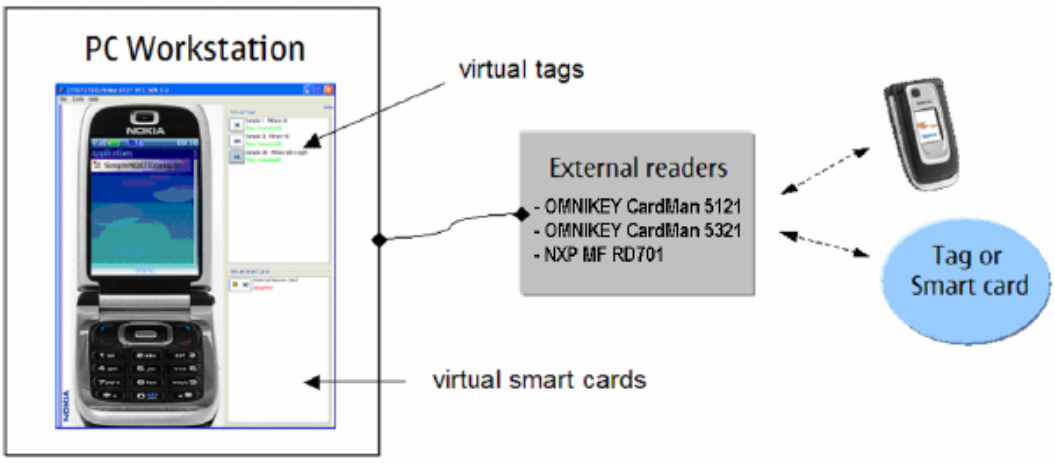


Figura 2.10: Simulador Nokia 6131

[19] JSR-257: Contactless Communication API, nos permite manejar el chip NFC por medio del lenguaje Java para teléfonos móviles con soporte Java.

El SDK de Nokia 6131 es una herramienta que permite la simulación y el desarrollo de aplicaciones NFC mediante el uso de teléfonos móviles de la serie 6131, brindando una interfaz gráfica, amigable con el usuario para una fácil relación con la tecnología.

Puede funcionar por si solo como un Kit de desarrollo o puede servir de soporte para ambientes de desarrollo integrados (IDE). Además ofrece soporte para MIDP 2.0, emulación del dominio de seguridad, emulación de Bluetooth, emulación de gráficos 3D, y emulación de las características de NFC. También proporciona soporte para la interfaz de emulación unificada (UEI)[20]. [25]

2.6.1.1. CAPACIDADES Y GENERALIDADES

Los requisitos mínimos para la instalación del SDK son:

- Sistema operativo Microsoft Windows XP con Service Pack 2
- Memoria RAM de 512 MB
- Procesador Pentium 1GHz
- Espacio Libre de Disco Duro de 200 MB

Estos requisitos fueron los que limitaron el uso por ejemplo de otro sistema operativo a la hora del desarrollo de la aplicación.

El paquete de instalación del SDK de Nokia incluye un *plugin* que permite el desarrollo de *MIDlets* tanto en *Netbeans*[21] como en *Eclipse*[22]. Además se puede utilizar su emulador para probar de una manera rápida aplicaciones de la plataforma Java sobre Teléfonos móviles (J2ME).

Por esta característica de soporte a IDEs[23], esta herramienta ha sido de mucha utilidad para realizar pruebas de la aplicación en *Netbeans* de este proyecto ya que a través de su simulación se pudieron verificar fallos y solucionarlos antes de la implementación y así tener una mayor eficiencia al desarrollarla.

[20] *http://java.sun.com/j2me/docs/uei_specs.pdf*
[21] *http://netbeans.org/*
[22] *http://www.eclipse.org/*
[23] IDE (integrated development environment), es un entorno de programación que ha sido empaquetado como un programa de aplicación.

Proporciona varias extensiones al estándar JSR-257. Estas extensiones permiten la aplicación de las comunicaciones P2P entre dos interfaces NFC y el protocolo (NFCIP-1) .Las etiquetas para lectura/escritura son: *mifare* 4k, *mifare* 1K, *mifare ultraligth*.

Las conexiones con las tarjetas soportadas por el estándar JSR-257 son:

- *TargetType.ISO14443_CARD* para ISO 14443-4 que complementa el acceso de las tarjetas inteligentes usadas en comando APDU[24].
- *TargetType.NDEF_TAG* para una tag que contiene datos formateados de NFC Forum.
- *TargetType.RFID_TAG* para tarjetas RFID en general.

2.6.1.2. TARJETA MIFARE 4K INTERNA Y ELEMENTO SEGURO

El elemento seguro consiste en una tarjeta inteligente Java y un área *Mifare* 4K para la emulación de la tarjeta. Se puede simular con el emulador de Nokia, pero esa emulación no contiene funcionalidad actual en el elemento seguro real.

En el emulador no se necesita ningún permiso para acceder al elemento seguro o para usar la conexión ISO14443, sin embargo si se quiere acceder al elemento seguro del teléfono, el *MIDlet* debe estar en la *Trusted 3rd party* del dominio de seguridad. Y esto se gestiona mediante emisores como *Venyon, Cassis* o *VivOtech*. Si el *MIDlet* no está en esa parte, saltara una excepción de seguridad al arrancar el *MIDlet*.

Cada dispositivo móvil contiene unas claves secretas para acceder al elemento de seguridad. Esta clave permite el libre acceso al elemento seguro y por lo tanto, debe mantenerse en secreto en todo momento. Esto permite por ejemplo instalar en el elemento seguro tarjetas de crédito, y nadie podrá tener acceso sin autorización.

Habrá un servicio disponible, que construya la conexión entre el elemento seguro y el servidor final. El servidor iniciado calcula la clave específica del chip ISD y establece un protocolo seguro de comunicación entre el elemento seguro y la parte final.

El servicio de desbloqueo de Nokia proporciona las claves de autenticación por defecto para instalar o quitar todas las aplicaciones de tarjetas inteligentes, *applets*, desde y hacia el elemento de seguridad del Nokia 6131 NFC.

[24] Application Protocol Data Unit, es la unidad de comunicación entre un lector de tarjetas inteligentes y una tarjeta inteligente.

Una vez desbloqueado el dispositivo, no se considera seguro por los emisores de aplicaciones de terceros. Por lo tanto no se podrá utilizar para instalar o ejecutar servicios de terceros que requieran una conexión segura y conexión de confianza entre el proveedor de servicios y el elemento de seguridad.

El desbloqueo de la tarjeta de emulación *Mifare* 4k contiene el valor por defecto en hexadecimal FFFFFFFFFFFF para ambas claves, *keyA* y *keyB*. Más adelante veremos la utilidad de estas claves en este proyecto.

Es útil conocer la estructura de la memoria interna de las etiquetas *Mifare*, para la funcionalidad del proyecto, porque podremos utilizar las interfaces eficazmente. Las claves de estas etiquetas se utilizan para proporcionar un alto nivel de seguridad. En consecuencia su memoria se organiza de una manera diferente que en las etiquetas *Mifare Ultralight.*

		Byte number within a block															
Sector	Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Description
0	0																	Manufacturer Data
	1																	Data
	2																	Data
	3	Key A						Access Bits				Key B						Sector Trailer 0
1	0																	Data
	1																	Data
	2																	Data
	3	Key A						Access Bits				Key B						Sector Trailer 1
..																		..

Figura 2.11: Estructura Tarjetas *Mifare*

La estructura de estas etiquetas se puede observar en la figura 3.2. Se componen de un total de 40 sectores. Los primeros 32 sectores contiene cada uno 4 bloques, mientras que los últimos 8 sectores se componen de 16 bloques cada uno de ellos. La longitud de cada bloque siempre son 16 bytes. [26]

En el sector 0, el bloque 0 llamado *Manufacturer Block* contiene datos especiales. Los cuatro primeros bytes contienen un identificador único (UID) de la tarjeta y a continuación un byte de verificación de conteo de bits (BCC) y el resto de bytes almacenan datos del fabricante. Este bloque está configurado y bloqueado para que no pueda modificarse. [27]

Figura 2.12: Bloque *Manufacturer*

Para poder realizar operaciones sobre los diferentes sectores de la tarjeta previamente hay que autenticarse. Se precisa conocer las claves *KeyA* y *KeyB* de cada sector. Si estas claves no se modifican se utiliza su valor por defecto mencionado anteriormente (FFFFFFFFFFFF). Estas claves están almacenadas en el último bloque de cada sector, llamado sector de remolque.

Figura 2.13: Sector de Remolque

La especificación para este tipo de tarjeta define seis operaciones que pueden ser usadas para acceder a los datos. Además, cada bloque de datos tiene tres bits que definen las condiciones de acceso. Estas condiciones definen la forma correcta de ejecutar cualquiera de las seis operaciones de memoria. Son los *Access Bits*.

Las condiciones de acceso para el sector de remolque, se especifican en la siguiente tabla:

Access bits			Access condition for					
			KEY A		Access bits		KEY B	
C1	C2	C3	read	write	read	write	read	write
0	0	0	never	key A	key A	never	key A	key A
0	1	0	never	never	key A	never	key A	never
1	0	0	never	key B	key A or B	never	never	key B
1	1	0	never	never	key A or B	never	never	never
0	0	1	never	key A	key A	key A	key A	key A
0	1	1	never	key B	key A or B	key B	never	key B
1	0	1	never	never	key A or B	key B	never	never
1	1	1	never	never	key A or B	never	never	never

Figura 2.14: Condiciones de acceso a los sectores

Con estas descripciones, para este proyecto simularemos las tarjetas del elemento seguro de cada dispositivo móvil con tarjetas externas *Mifare* 4K a las que si podremos tener acceso. Además para dar una mayor seguridad a la aplicación, se modificaran estas claves. Por lo que si no se conocen no se podrá modificar los datos que se encuentran en las tarjetas.

En la tabla, vemos los valores que deberemos dar a los *Access Bits*, para poder leer y escribir en los diferentes sectores.

Life Cycle State		MAD1 and MAD2 Sectors	NFC Forum Sectors	MAD1 and MAD2 Sector Trailer Bytes Values			NFC Forum Sector Trailer Bytes Values		
	Access Bits	Access Bits Values	Access Bits Values	Byte 6	Byte 7	Byte 8	Byte 6	Byte 7	Byte 8
INITIALIZED and READ/WRITE	$C1_0$ $C2_0$ $C3_0$	100b[1]	000b	78h	77h	88h	7Fh	07h	88h
	$C1_1$ $C2_1$ $C3_1$	100b	000b						
	$C1_2$ $C2_2$ $C3_2$	100b	000b						
	$C1_3$ $C2_3$ $C3_3$	011b	011b						
READ-ONLY	$C1_0$ $C2_0$ $C3_0$	010b[1]	010b	07h	8Fh	0Fh	07h	8Fh	0Fh
	$C1_1$ $C2_1$ $C3_1$	010b	010b						
	$C1_2$ $C2_2$ $C3_2$	010b	010b						
	$C1_3$ $C2_3$ $C3_3$	110b	110b						

Figura 2.15: Condición de los *Access Bits* [28]

Como se ve, hay dos tipos de sectores:

- Sectores MAD (*Mifare Application Directory*): Este sector se utiliza para identificar que sectores se dedican a una aplicación específica. Directorio de aplicaciones, que permiten saber donde esta una aplicación concreta sin necesidad de realizar una búsqueda por toda la tarjeta.
- Sectores NFC Forum: Contienen datos definidos en el NFC Forum.

CAPÍTULO 3

APIs

Este capítulo expone el conjunto de APIs empleadas para el desarrollo de las aplicaciones que hace uso de la tecnología NFC y Bluetooth.

Se detallan las APIs empleadas, JSR 257 y JSR 82, explicando el funcionamiento de los paquetes utilizados.

3.1 API NFC JSR 257

El API para Comunicaciones *Contactless* está basado en la especificación JSR-257 de Java y se compone de dos fases, el descubrimiento e intercambio de datos entre etiquetas NFC, etiquetas RFID y *Smart Cards* mediante el uso de paquetes o librerías que están definidas de acuerdo a la función que se desea realizar.

Este API opcional del J2ME consiste en una interfaz de programación que permite a las aplicaciones acceder a información en *contactless targets*. A continuación se describe su estructura y las clases principales de cada uno de los paquetes que la componen.

Las clases e interfaces se dividen en cinco paquetes:

- *javax.microedition.contactless*: Provee descubrimiento de *target* y clases comunes a todos esos *targets.*
- *javax.microedition.contactless.ndef*: Contiene clases e interfaces necesarias para la comunicación con *tags* que tienen datos en formato NDEF.
- *javax.microedition.contactless.rf*: Permite comunicación con *tags* RFID que no tienen datos en formato NDEF.
- *javax.microedition.contactless.sc*: Habilita la comunicación con *smart cards*, tanto con la interna del dispositivo móvil como con las tarjetas externas compatibles con el estándar ISO 14443.
- *javax.microedition.contactless.visual*: Permite la lectura y generación de *tags* visuales. Se soportan los tipos más comunes de códigos de barras y matriciales como el *DataMatrix* o el *QR Code*.

En este proyecto, se ha utilizado el primer paquete *javax.microedition.contactless*, que se detalla más ampliamente a continuación.

Paquete *javax.microedition.contactless*:

Contiene las clases e interfaces necesarias para el descubrimiento del objetivo de contacto. El descubrimiento de *contacteless targets* es el punto de partida de este API. Cuando se descubre un *target*, la aplicación podrá comunicarse con él. Entre las clases e interfaces encontramos:

- Clase *DiscoveryManager*: Ofrece mecanismos para el descubrimiento de *contactless targets* y maneja los diferentes tipos de *listeners* para este API.
- Interfaz *TagConnection*: Toda conexión que pueda ser usada para la comunicación con *tags* RFID o *smart cards* externos debe extender esta interfaz.
- Interfaz *TargetListener*: Provee mecanismos para que la aplicación sea notificada cuando el hardware (teléfono móvil) descubre un *contactless targets*.
- Interfaz *TargetPropierties*: Recoge las propiedades que son comunes a todos los *contactless targets* soportados por esta especificación como: mapping, uid o url.
- Clase *TargetType:* Recoge todos los tipos de *contactless targets* soportados por este API como: ISO14443_CARD, NDEF_TAG, RFID_TAG o VISUAL_TAG.
- Interfaz *TransactionListener*: Provee la notificación a la aplicación cuando hay actividad del *secure element*, cuando el móvil esta en modo de emulación de tarjetas.
- Clase *ContactlessException:* produce excepción cuando una operación no admitida se intenta.

3.1.1. EXTENSIONES DEL API JSR 257

Además el API JSR-257 contiene las extensiones específicas de API que proporciona interfaces de comunicación y acceso a determinados tipos de objetos.

El SDK de Nokia 6131 NFC soporta todas estas extensiones, pero algunas de las aplicaciones implementadas con estas extensiones deben probarse y ejecutarse en un dispositivo real. [29]

Para el desarrollo de este proyecto se emplearon dos extensiones que proporciona el API JSR 257:

- *com.nokia.nfc.nxp.mfstd*: Proporciona las clases e interfaces necesarias para la comunicación con una tarjeta *Mifare* estándar.
- *com.nokia.nfc.p2p*: Proporciona un interfaz para la comunicación de dispositivos a través de NFCIP.

A continuación se detalla los tipos de comunicaciones empleadas en la implementación del proyecto que requieren la utilización de estos dos paquetes.

MFSTANDARDCONNECTION

Proporciona un punto de partida para comunicarse con las tarjetas *Mifare* 1K y 4K. Esta interfaz está dentro del paquete *com.nokia.nfc.nxp.mfstd*. La figura 3.7 muestra como las clases e interfaces se relacionan entre sí.

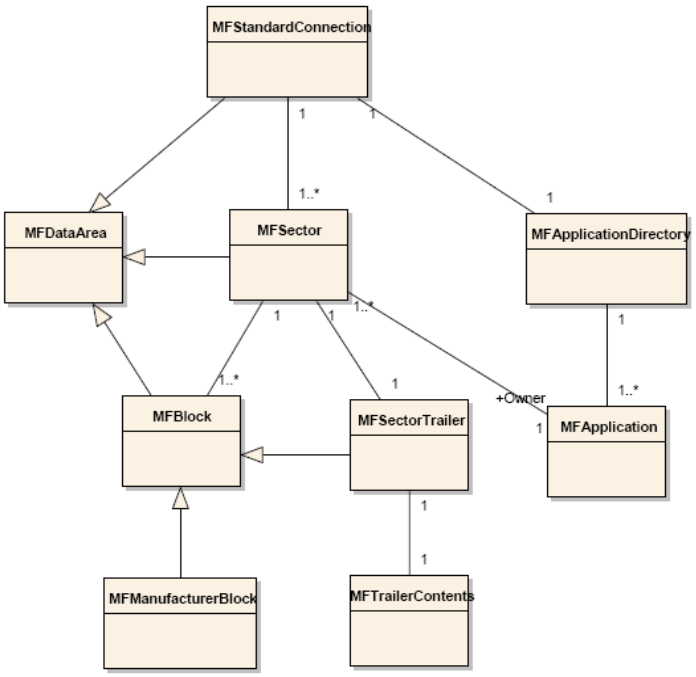


Figura 3.1: Diagrama de Clases de la API *Mifare Standard*

De todas estas clases, para la implementación de la comunicación con la tarjeta *Mifare* en el proyecto se utilizaron las siguientes:

- Interfar *MFStandardConnection*: Esta interfaz representa la conexión a las tarjetas estándar *Mifare* 1K y 4K.
- Interfaz *MFBlock*: Interfaz para el manejo de bloques de las etiquetas estándar *Mifare*.

- Interfaz MFManufacturerBlock: Representa el interfaz para acceder al bloque del fabricante de las etiquetas estándar *Mifare*.
- Interfaz MFSector: Proporciona el acceso a los sectores dentro de las etiquetas estándar *Mifare*.
- Interfaz MFSectorTrailer: Proporciona el acceso al sector de remolque.
- Clase MFAccessBits: Clase para la manipulación de los bits de acceso de las etiquetas estándar *Mifare.*
- Clase MFKey: Clase que representa la clave de identificación de las etiquetas.
- Clase *MFKey.KeyA*: Clase que representa el tipo de autenticación de la clave A de las etiquetas estándar *Mifare.*
- Clase *MFKey.KeyB*: Clase que representa el tipo de autenticación de la clave B de las etiquetas estándar *Mifare.*
- Clase MFTrailerContents: Describe el contenido del sector de remolque, incluyendo las claves de autenticación y los bits de acceso.
- Clase MFStandardException: Se lanza para indicar un error especifico de la comunicación *Mifare Standard*, es decir, si la autenticación falla o si las condiciones del método no se cumplen.

Este paquete nos permite en el desarrollo del proyecto, cambiar las claves de las etiquetas *Mifare* para proporcionarles seguridad, y la escritura y lectura de los datos personales de cada empleado. Las aplicaciones que implementan estas capacidades estarán contenidas tanto en el teléfono móvil del Empleado, para poder escribir y leer los datos personales de su propia tarjeta, como en el teléfono móvil del Vigilante y el teléfono móvil que actúa como Lector, para poder leer los datos personales de las tarjetas.

NFCIPCONNECTION

NFCIP permite a dos dispositivos comunicarse entre sí cuando están dentro del rango (próximos). La comunicación sigue un protocolo simple de solicitud- respuesta. La interfaz *NFCIPConnection* se encuentra dentro del paquete *com.nokia.nfc.p2p* como se vio anteriormente.

La comunicación *peer to peer* no está contemplada en el JSR 257, porque no está estandarizada en el NFC Forum, sin embargo Nokia lo incorpora como una extensión al API.

En este tipo de comunicación, un *MIDlet* puede funcionar como *Initiator* o como *Target*. El *Initiator* es el que establece la conexión, envía algunos datos y espera a recibir respuesta. Mientras que el dispositivo *Target* espera a recibir datos para poder enviar una respuesta.

La capacidad que nos ofrece este tipo de comunicación se utiliza en el desarrollo del proyecto para la comunicación entre los dispositivos móviles que intervienen en el sistema: un teléfono móvil del Empleado, y un teléfono móvil del Vigilante. Esto les permite intercambiar, mediante este mecanismo de "enviar-recibir", toda la información relacionada con los datos personales de cada Empleado.

3.2. API BLUETOOTH JSR-82

En las comunicaciones Bluetooth hay un Servidor, que es el dispositivo que ofrece un servicio y un Cliente, que es el dispositivo que accede a él. Cada desarrollador puede definir aplicaciones propias de servidor Bluetooth más allá de los especificados en los perfiles de Bluetooth y hacer que estos servicios estén disponibles para los clientes remotos. Para ello se define un servicio de registro (*service record*) que describa ese servicio.

En la parte de la comunicación servidora debemos crear una conexión servidora, especificar los atributos del servicio y abrir las conexiones del cliente.

Mientras que en la comunicación cliente deberá realizar la búsqueda de dispositivos, para realizar después la búsqueda de servicios, establecimiento de la conexión, una vez encontrado el dispositivo y la comunicación con éste.

Como se vio en el capítulo 2, en la sección "Aplicaciones con Java" el API JSR 82 permite a *MIDlets Java*, usar Bluetooth en sus dispositivos. Además cuenta con una librería basada en esta implementación, ***Bluecove***, que puede ser usada en la *Java Standard Edition* (J2SE) 1.1 o superiores, y está habilitada para varios sistemas operativos.

En este proyecto, la comunicación Bluetooth se ha llevado a cabo entre los dispositivos móviles del Vigilante y Lector Fijo y el PC.

Los dispositivos móviles a través de J2ME utilizaran el API JSR 82, funcionando como clientes, mientras que el PC a través de J2SE utilizara la implementación del API JSR 82 desarrollada por *Bluecove*, funcionando como servidor.

El API JSR 82 cuenta con dos paquetes esenciales, *javax.bluetooth* y el *javax.obex.* Para la implementación de la comunicación Bluetooth se utiliza el primer paquete. A continuación se detallan las clases e interfaces de este paquete:

3.2.1. PAQUETE JAVAX.BLUETOOTH

Interfaz DiscoveryListener

Permite que una aplicación especifique un *listener* que escuche y notifique eventos a la aplicación cada vez que se descubre un dispositivo, un servicio o se finaliza una búsqueda.

Esta interfaz proporciona cuatro métodos, dos para descubrir dispositivos y dos para descubrir servicios.

El método *deviceDiscovered()*, se llama cuando se encuentra un dispositivo durante la búsqueda. Una vez finalizada la búsqueda de dispositivos porque se ha completado o se ha cancelado se llama al método *inquiryCompleted().*

Mientras que para la búsqueda de servicios se emplea el método *servicesDiscovered()*, y una vez completada esa búsqueda se hace una llamada al método *servicesSearchCompleted().*

Clase LocalDevice

Representa al dispositivo local, y permite obtener información sobre el dispositivo: modo de conectividad, dirección Bluetooth y nombre. Se debe tener una única instancia de esta clase, para obtenerla se utiliza el método *getLocalDevice().* Para definir la visibilidad del dispositivo se emplea el método *setDiscoverable(int mode)*. Las opciones de visibilidad que hay son:

- DiscoveryAgent **GIAC** (*General/Unlimited Inquiry Access Code*): Conectividad ilimitada.
- DiscoveryAgent **LIAC** (*Limited Dedicated Inquiry Access Code*): Conectividad limitada.
- DiscoveryAgent **NOT_DISCOVERABLE**: El dispositivo no será visible para el resto de dispositivos.

Clase DeviceClass

Tiene los métodos para recuperar los valores de las clases que describen las propiedades de un dispositivo. Describe el tipo de dispositivo que es.

Clase DiscoveryAgent

Esta clase proporciona métodos para el descubrimiento de dispositivos y de servicios. Este objeto es único y se obtiene a través del método *setDiscoveryAgent()* del objeto *LocalDevice.*

Para el descubrimiento de dispositivos, se emplean dos métodos, *starInquiry()* que inicia una investigación para encontrar dispositivos y *retrieveDevice()* que devuelve los dispositivos encontrados en búsquedas anteriores. También proporciona una manera de cancelar una búsqueda mediante el método *cancelInquiry()*.

Para el descubrimiento de servicios, se cuenta con dos métodos, *searchServices()* para buscar un servicio en un dispositivo único, y *selectService()*, que busca el servicio en un conjunto de dispositivos remotos.

Clase UUID

Esta clase encapsula enteros sin signo que son de 16 bits, 32 bits o 128 bits de longitud. Representa a un identificador único universal utilizado como valor para un atributo de un servicio. Solo los atributos de servicio representados por un UUID son buscado por el Bluetooth SDP (*Service Discovery Protocol*).

Clase DataElement

Contiene los distintos tipos de datos que un valor de atributos de servicio Bluetooth puede asumir. Pueden ser enteros con o sin signo, cadenas de texto, booleanos UUID o una secuencia de cualquiera de estos tipos de datos.

Interfaz ServiceRecord

Describe las características de un servicio Bluetooth. Se compone de atributos de servicio, donde cada atributo es un par ID y un valor de atributo. El ID es un entero sin signo.

El *ServiceRecord* se obtiene a través de la clase *LocalDevice* con el método *getRecord().*

Para establecer sus atributos utilizamos el método *setAttributeValue()* al que le pasaremos el identificador numérico y un objeto *DataElement*. [30]

CAPÍTULO 4

DESCRIPCIÓN FUNCIONAL

Para el desarrollo del proyecto se han seguido tres pasos, especificación de los requisitos, desarrollo de la aplicación y pruebas del sistema.

Este capítulo explica el conjunto de requisitos y objetivos propuestos que debe cumplir la aplicación, para que el desarrollo sea más sencillo y correcto en el menor tiempo posible. Así como el diseño de cada una de las partes que componen el sistema final

4.1. ESPECIFICACIÓN

En este proyecto se ha implementado un conjunto de aplicaciones que permiten realizar el control de acceso a las diferentes zonas con las que cuenta un hotel y un sistema de identificación para los empleados.

En este apartado se intenta dar una visión global de los requisitos del proyecto para su posterior implementación. El sistema final cuenta con 5 módulos diferenciados, cada uno de los cuales se implementa con una aplicación diferente. Además se han desarrollado dos aplicaciones adicionales para pruebas de funcionamiento y uso de los trabajadores del hotel que se encargan de la gestión de los empleados.

En cada uno de estos módulos la funcionalidad y los requisitos son diferentes. Veremos detalladamente cada uno de estos módulos y las dos aplicaciones adicionales implementadas.

4.1.1. FUNCIONALIDAD

A continuación presentamos la funcionalidad de los diferentes módulos para tener una visión global de todo el sistema, comenzando por las dos aplicaciones adicionales.

4.1.1.1. APLICACIONES ADICIONALES

Una de las aplicaciones adicionales al resto de módulos, permite cambiar las claves de las tarjetas *Mifare* que llevarán los empleados en sus dispositivos móviles, mientras que la otra permite guardar los datos personales de cada empleado antes de entregar los dispositivos a cada uno de ellos.

Aplicación *SecureWriterMF*

Las etiquetas internas del elemento seguro de cada dispositivo se simulan con tarjetas externas *Mifare* 4K. Esta aplicación permite cambiar las claves de estas tarjetas para proporcionarles seguridad.

Aplicación *AccessSecureTag*

Cada empleado dispone de un dispositivo móvil con sus datos personales guardados en la etiqueta interna del elemento seguro del dispositivo. Como se ha comentado estas etiquetas se simulan con tarjetas externas *Mifare* 4K.

Antes de proporcionarle a cada empleado su dispositivo móvil, las etiquetas deben contener ya sus datos personales. Esta aplicación permite guardar esos datos. Además se podrá utilizar para pruebas del sistema, tanto de lectura de datos de las tarjetas como escritura.

4.1.1.2. MÓDULO EMPLEADO

Cada empleado dispone de un dispositivo móvil con tecnología NFC (Nokia 6131) y una tarjeta externa Mifare. Esta tarjeta simula la etiqueta del elemento seguro del dispositivo al que no podemos acceder como se explicó en el Capítulo 1 en la sección 2.6.1.2. “Tarjeta *Mifare* 4K Interna y Elemento Seguro”. En ella están contenidos los datos personales, y será el propio dispositivo del empleado el único que pueda cambiar esos datos (simulando que es el elemento seguro).

Si se produce un cambio en la BBDD sobre algún dato personal del empleado, éste deberá modificarlos en la tarjeta. Se comunicará mediante *peer to peer* con el módulo Vigilante, que es el encargado de comunicarle los nuevos datos.

Este módulo es una aplicación que permite:

- Escritura en la tarjeta *Mifare* mediante comunicación NFC.
- Comunicación *peer to peer* con el módulo Vigilante.

4.1.1.3. MÓDULO VIGILANTE

Existe un empleado de Seguridad que dispone de un teléfono móvil Nokia 6131 y una tarjeta externa *Mifare*. Al ser un empleado, en la tarjeta están contenidos sus datos personales, como en el módulo anterior.

Comprueba que el resto de empleados tiene permisos para estar en una zona determinada. Esta comprobación se realiza mediante la lectura de los datos del empleado y la consulta al servidor mediante la tecnología Bluetooth. Si detecta que se ha producido algún cambio en los datos del empleado, le comunica los nuevos datos mediante *peer to peer.*

Este módulo es una aplicación que permite:

- Lectura de datos mediante comunicación NFC.
- Comunicación *peer to peer* con el módulo Empleado.
- Comunicación Bluetooth con el Servidor Bluetooth.

4.1.1.4. MÓDULO LECTOR FIJO

A la entrada de cada zona se dispone de un Lector Fijo NFC. Este Lector permite la entrada a la zona a los empleados que tienen permisos para acceder. Cuando un dispositivo se acerca al Lector, éste lee el identificador único de la tarjeta *Mifare,* y se comunica con el Servidor para comprobar los permisos.

Como no se dispone de Lectores NFC para la realización del proyecto, esta funcionalidad se implementa mediante un teléfono móvil Nokia 6131.

El lector lee los datos de las tarjetas de los empleados que intenten acceder a la zona y a continuación se comunica con el servidor Bluetooth para comprobar los permisos del empleado. Y en función de éstos le permitirá el paso o se lo denegará.

Este módulo es una aplicación que permite:

- Lectura de datos mediante la comunicación NFC.
- Comunicación Bluetooth con el Servidor Bluetooth.

4.1.1.5. MÓDULO SERVIDOR BLUETOOTH

El servidor Bluetooth atiende las consultas de los clientes, Lector Fijo y el Vigilante. Se comunica la BBDD para obtener la información requerida en las consultas. Obtiene los datos personales de los empleados y las zonas de acceso de los mismos.

Si el empleado que accede a una zona es un Vigilante de Seguridad, este módulo se encargará de guardar y actualizar el historial para dicho empleado.

Estos historiales permiten tener un control sobre las rutas seguidas por los Vigilantes de Seguridad en sus rondas de vigilancia. Día, zonas a las que acceden y horas de entrada y salida.

Este módulo es una aplicación que permite:

- Comunicación con los clientes mediante la tecnología Bluetooth.
- Consultas a la BBDD.

4.1.1.6. MÓDULO RED HOTEL

Este módulo se compone del Servidor donde se aloja la aplicación web del Hotel "Gestión Empleados", de la BBDD de los empleados y de la aplicación web que reside en el servidor.

La BBDD contiene los datos personales de cada empleado, las zonas de acceso a las que cada uno puede entrar y el historial de los diferentes Vigilantes de Seguridad.

Mientras que la aplicación Web nos permite, modificar los datos de cada empleado, ya sean datos personales o zonas de acceso, añadir y borrar empleados y tener un seguimiento del historial de cada Vigilante de Seguridad.

Este módulo es una aplicación que permite:

- Gestión de los datos personales de cada empleado mediante conexión a la BBDD.
- Seguimiento del historial de los Vigilantes de Seguridad mediante conexión a la BBDD.

4.1.2. REQUISITOS

Para mejorar la funcionalidad, y sacar el máximo partido a la aplicación se llevo a cabo un análisis previo, en el que se definieron una serie de requisitos que deben cumplirse.

- Darle seguridad a las etiquetas *Mifare* 4K.

- Guardar de forma segura la información personal en cada etiqueta.
- Leer de forma segura la información personal en cada etiqueta.
- Transmisión correcta entre dos dispositivos mediante NFCIP.
- Comunicación Bluetooth. El servidor puede atender a varios clientes a la vez.
- Permitir o denegar la entrada a una zona.
- Seguimiento de las zonas a las que accede el Vigilante. Historial.
- Modificación de las zonas a las que un empleado tiene acceso.
- Añadir empleados nuevos y borrar empleados ya existentes.

4.2. DISEÑO

4.2.1. DISEÑO DE BLOQUES

Ahora que se tiene claro que es lo que hay que hacer, el siguiente paso es realizar un diseño de alto nivel de lo que se va a implementar. Primero se diseña un esquema de bloques para cada módulo indicando las aplicaciones, con sus componentes y las relaciones entre ellas.

La idea principal del proyecto es tener un control total sobre los empleados de un hotel. Se podrán gestionar los datos personales mediante una página web, y las zonas a las que cada empleado puede acceder. Estas zonas están definidas en la base de datos, y los permisos de acceso pueden modificarse según se requiera.

Cada empleado dispone de un dispositivo móvil con tecnología NFC con sus datos personales, que debe acercar al lector de la zona cuando quiera entrar. Estos datos están guardados de forma segura. De modo que si no se conocen las contraseñas correctas, no se podrán modificar los datos ni leer dicha información.

El lector colocado a la entrada de cada zona permitirá el paso de los empleados que tengan permisos para acceder a dicha zona, denegándoselo al resto.

El Vigilante de Seguridad realiza rondas por todo el hotel, vigilando que los empleados puedan estar en las distintas zonas. Además será el encargado de informar a los empleados de que sus datos personales se han modificado cuando esto ocurra.

A continuación se explica el diseño de cada uno de los módulos explicados anteriormente, y posteriormente se realizará un análisis de las comunicaciones entre los distintos módulos.

4.2.1.1. MÓDULO EMPLEADO

Este módulo representa la figura del Empleado, se compone de un dispositivo móvil y una tarjeta *Mifare* 4K.

Cuando el empleado quiere acceder a una zona el Lector Fijo, simulado con un teléfono móvil, leerá los datos de la tarjeta mediante la comunicación NFC. De la misma forma cuando un Vigilante de Seguridad quiere conocer los datos del empleado, acercará su terminal a la tarjeta.

Para este tipo de comunicación el módulo Empleado no necesita implementar nada, ya que la lectura de los datos se implementa en los otros dos módulos.

Figura 4.1: Estructura a generar del módulo Empleado, NFC

Cuando el Vigilante de Seguridad detecta que algún dato de la tarjeta del empleado no coincide con los almacenados en la BBDD, por que se ha modificado, le informa al Empleado que debe realizar la actualización de sus datos. Para ello el Empleado debe arrancar en su dispositivo la aplicación *MIDlet* llamada "Empleado".

Esta aplicación permite tener una comunicación *peer to peer* con otro dispositivo, para la recepción de los nuevos datos que se deben guardar y una comunicación NFC con la propia tarjeta del módulo Empleado, para guardar esos datos.

La actualización de los datos se realiza de la siguiente forma:

1. El Vigilante informa al Empleado de que los datos no coinciden con los almacenados en la BBDD y debe cambiarlos.
2. El Empleado arranca la aplicación *MIDlet* llamada “Empleado”.
3. El Empleado activa la comunicación NFCIP para la recepción de los datos.
4. El Vigilante activa su aplicación en modo conexión NFCIP.
5. Se acercan los dispositivos, y se produce el intercambio de información.
6. El Empleado activa la comunicación NFC para guardar esos datos en la Tarjeta.
7. Guarda los datos en la Tarjeta.

Figura 4.2: Estructura a generar del módulo Empleado, NFCIP

4.2.1.2. MÓDULO VIGILANTE

El Vigilante de Seguridad podrá comprobar los datos y las zonas de acceso de cada empleado. Para ello el Vigilante debe arrancar en su dispositivo la aplicación *MIDlet* llamada “Vigilante”.

Esta aplicación permite leer datos de las tarjetas *Mifare* de los empleados mediante una comunicación NFC. También permite recibir información del módulo Servidor Bluetooth, mediante la comunicación Bluetooth. Y por último permite la comunicación *peer to peer* con el módulo empleado como se comentó antes.

Figura 4.3: Estructura a generar del módulo Vigilante

La comprobación de los datos del empleado y las zonas de acceso se realiza de la siguiente forma:

1. El dispositivo Vigilante arranca la aplicación *MIDlet* llamada “Vigilante”.
2. Se acerca a la tarjeta del empleado.
3. Se produce la lectura de los datos mediante la comunicación NFC.
4. El Vigilante activa en la aplicación la comunicación Bluetooth y busca al servidor.
5. Una vez conectado, le pide la información del empleado (datos personales y zonas de acceso).
6. Cuando tiene esa información, la aplicación muestra por pantalla, la información de la tarjeta del empleado y la información proporcionada por el servidor.
7. Comprueba que puede estar en la zona.
8. Comprueba si se hay diferencias entre los datos de la tarjeta y los datos proporcionados por el servidor.
9. Si hay diferencias, el Vigilante avisa al Empleado que debe modificar sus datos.

Figura 4.4: Estructura a generar del módulo Vigilante, NFCIP

Como se ha mencionado en el punto anterior, la transmisión de los nuevos datos se realiza mediante la comunicación *peer to peer*, explicada en ese punto.

4.2.1.3. MÓDULO LECTOR FIJO

La estructura deseada de este módulo es colocar un Lector Fijo a la entrada de cada zona. Estos lectores se comunican con el servidor mediante conexión por cable, como se muestra en la figura 4.5, para que las consultas a la BBDD y respuestas fuesen mucho más rápidas y seguras.

Figura 4.5: Estructura deseada del módulo Lector Fijo

Como no contamos con Lectores NFC, la implementación del Lector, se ha simulado con un dispositivo móvil NFC, Nokia 6131, como se observa en la figura 4.6, la conexión con el servidor se realiza con la tecnología Bluetooth.

Figura 4.6: Estructura a generar del módulo Lector Fijo

Como se ha comentado, cuando un empleado quiere acceder a una zona, acerca su tarjeta al lector y éste se encarga de comprobar si tiene permisos para entrar o no. Para ello el dispositivo que simula al Lector Fijo debe arrancar la aplicación *MIDlet* llamada "LectorFijoZona*", en función de la zona a la que de acceso.

Esta aplicación permite leer los datos de la tarjeta del empleado mediante comunicación NFC, y la recepción de la información del módulo Servidor Bluetooth, mediante comunicación Bluetooth.

Para comprobar si el empleado tiene permiso para acceder a la zona se siguen los siguientes pasos:

1. El dispositivo Lector Fijo debe tener arrancada la aplicación *MIDlet* llamada "LectorFijoZona*".
2. Esta aplicación esta en modo comunicación NFC, a la espera de que se le acerque alguna tarjeta.
3. El empleado acerca su tarjeta al lector.
4. El Lector lee los datos, y arranca la comunicación Bluetooth y busca al servidor.
5. Una vez conectado, le pide información sobre el empleado (zonas de acceso).
6. Cuando dispone de esa información, muestra por pantalla un mensaje, según si el empleado tiene permiso para entrar o no.

4.2.1.4. MÓDULO SERVIDOR BLUETOOTH

Como se ha visto, tanto el módulo Vigilante como el módulo Lector Fijo se comunican con éste para obtener información sobre un empleado. Este paso de información se realiza con la comunicación Bluetooth. Para ello debe tener arrancada una aplicación en el servidor llamada “ServidorBluetooth”.

Esta aplicación permite comunicarse y cambiar información con los módulos Lector Fijo y Vigilante, mediante la comunicación Bluetooth. Puede establecer varias comunicaciones Bluetooth a la vez. Además permite realizar consultas a la BBDD, mediante la conexión con ella (implementada en MySQL) a través del driver JDBC (*Java Data Base Connectity*) instalado en el módulo red hotel.

Figura 4.7: Estructura a generar del módulo Servidor Bluetooth

Para transmitir información a los módulos Lector Fijo o Vigilante se siguen los siguientes pasos:

1. El servidor Bluetooth debe tener arrancada la aplicación llamada “ServidorBluetooth”.
2. Esta aplicación espera recibir peticiones de conexión Bluetooth. Por cada petición abre una comunicación.
3. Cuando se establece la comunicación, se espera a recibir la consulta por parte del módulo Lector Fijo o Vigilante.
4. Estos módulos realizan las consultas comentadas en los apartados anteriores.
5. La aplicación se conecta a la BBDD mediante el driver JDBC y realiza las consultas necesarias.
6. Le envía la información pedida a los módulos, y cierra las conexiones.

4.2.1.5. MÓDULO RED HOTEL

Este módulo representa la red de un hotel a través de la cual se puede acceder a los datos personales de los empleados, las zonas de acceso y el historial de los vigilantes, compilados en una BBDD. A través de la aplicación Web “GestionEmpleados”.

Esta aplicación permite gestionar los datos de los empleados, añadiendo empleados nuevos a la BBDD o borrando empleados, así como modificando sus datos personales o las zonas de acceso. Permite también llevar un control sobre las rutas seguidas por los vigilantes de seguridad en sus rondas (Historial). Para todo esto se ha desarrollado una página web.

Figura 4.8: Estructura a generar del módulo Red Hotel

Los empleados encargados del personal del hotel, acceden a través de su ordenador en su puesto de trabajo a la aplicación web “GestionEmpleados”. A través de esa aplicación, se podrán realizar todas las gestiones antes mencionadas.

Tanto la aplicación “ServidorBluetooth” del módulo Bluetooth como la aplicación “GestionEmpleados” residen en la misma máquina de la red del Hotel.

El modelo Relacional de la BBDD se muestra en la figura 4.9.

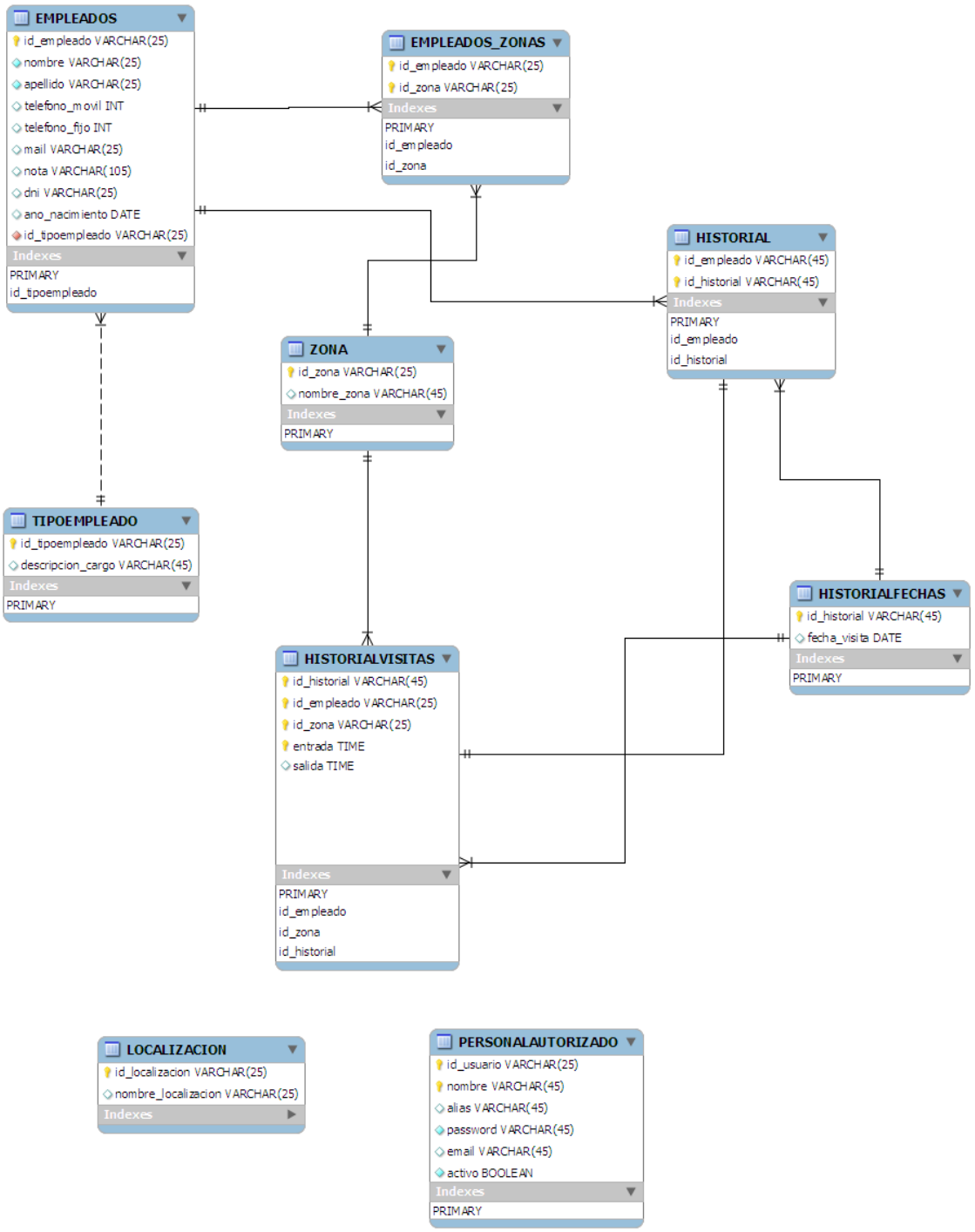


Figura 4.9: Modelo Relacional

PersonalAutorizado

La entidad PersonalAutorizado se utiliza para almacenar la información referente a los empleados que tienen acceso a la gestión de los empleados y zonas del Hotel.

El atributo **activo**, describe si el empleado puede o no acceder a la gestión del Hotel, aunque este registrado.

Empleados y TipoEmpleado

La entidad Empleados se usa para almacenar los datos personales de los empleados del hotel. Cuenta con diez propiedades que intentan recoger los datos más relevantes de un empleado. El atributo **Id_empleado** es el UID de las tarjetas *Mifare* de cada empleado.

La entidad TipoEmpleado define el cargo de cada empleado. Tenemos diez cargos definidos para los empleados del Hotel en este proyecto.

Id_tipoEmpleado	Descripción_Cargo
grupoA	Limpieza
grupoB	Cocina
grupoC	Camarero
grupoD	Recepción
grupoE	Transportista
grupoF	Gerente
grupoG	Servicio Técnico
grupoH	Botones
grupoI	Mantenimiento
grupoJ	Seguridad

Tabla 4.1: Tipos de Empleados

Zona Y Empleados_Zonas

Las diferentes zonas del Hotel, las hemos definido con un identificador único mediante el atributo **Id_zona.** En el proyecto se decidió definir únicamente siete zonas, descritas en esta entidad.

Id_Zona	Nombre_Zona
zonaA	Cocina, comedor, almacén de comida
zonaB	Office
zonaC	Habitaciones
zonaD	Recepción
zonaE	Despachos
zonaF	Almacén de Servicios
zonaG	Descarga

Tabla 4.2: Zonas

La entidad Empleados_Zonas define las zonas a las que cada empleado puede acceder.

Historial, HistorialFechas e HistorialVisitas

Los empleados "vigilantes", tendrán un historial abierto. Cada historial se identifica mediante el atributo **Id_historial**. Además cada día se abrirá un historial diferente.

La entidad HistorialVisitas define la zona a la que accede el vigilante, así como la hora de entrada y de salida de la zona.

4.2.2. DISEÑO LÓGICO

Una vez se han explicado los diferentes módulos y la relación entre ellos, se explicará más a fondo las diferentes comunicaciones que forman parte de estos módulos.

4.2.2.1. COMUNICACIÓN NFC

Esta comunicación se establece en las dos aplicaciones adicionales implementadas. En la primera se utiliza para el cambio de claves, y en la segunda para guardar los datos en las tarjetas y leerlos.

En los módulos Empleado, Vigilante y Lector Fijo, también se emplea esta comunicación, que seguirá el mismo funcionamiento que para la segunda aplicación adicional.

Cambio de claves de las tarjetas

La aplicación ***SecureWriterMF***, se encarga de cambiar las claves de las tarjetas para darles seguridad y que nadie pueda leer los datos ni guardar información si no se conocen las claves.

Las claves se encuentran en el último bloque de cada sector llamado, sector de remolque. Estas claves son la *KeyA*, los *AccessBits* y la *KeyB*. Como se vio en el Capítulo 2 en la sección "Elemento seguro y tarjeta *Mifare* 4K Interna".

Esta aplicación es un simple *MIDlet* J2ME, que cambia el valor del contenido de los sectores de remolque de los sectores donde desee escribir. Cuando el *MIDlet* se inicia, únicamente hay que acercar la tarjeta para que sus claves se cambien.

Escritura y Lectura Segura

La aplicación *AccessSecureTag* escribe los datos personales en las tarjetas antes de que el empleado disponga del dispositivo con la tarjeta.

Los datos personales de cada empleado que se guardan en las tarjetas son:

Nombre, Apellido, Teléfono Móvil, Teléfono Fijo, Mail y Nota.

La aplicación escribe en las tarjetas *Mifare* y lee la información contenida en ella, se utiliza inicialmente para guardar la información personal de cada empleado en sus respectivos dispositivos, y para pruebas iniciales, como ya se comentó.

Tiene cuatro opciones a escoger, dos que simulan la lectura y escritura de la etiqueta del elemento seguro del dispositivo móvil, y dos que ejecutan la lectura y escritura de la tarjeta *Mifare* 4K externa.

Las dos opciones de simulación se desarrollaron con vista a un futuro, cuando sea posible colocar el *MIDlet* en la *Trusted 3rd party* del dominio de seguridad. Por tanto estas opciones solo se han podido probar en el simulador SDK Nokia 6131. Las otras dos opciones son las que utilizaremos para escribir y leer información en las tarjetas *Mifare*.

Esta aplicación se integra dentro de la aplicación del módulo Empleado. La opción de lectura de la tarjeta *Mifare* 4K externa se integra dentro de las aplicaciones de los módulos Lector Fijo y Vigilante.

4.2.2.2. COMUNICACIÓN NFCIP

Esta comunicación se establece entre los módulos Vigilante y Empleado, para el intercambio de los datos del empleado actualizados. Funciona de la siguiente forma:

Ambos dispositivos cuando activan esta comunicación están en **modo activo**, que como detallamos en capítulos anteriores, implica que los dos dispositivos generen su propio campo de radiofrecuencia, desactivando ese campo mientras espera respuesta.

La aplicación “Empleado” será el dispositivo *Target*, esperando el inicio de la comunicación por parte del dispositivo *Initiator* que será la aplicación “Vigilante”. Éste le envía los datos mediante la conexión *peer to peer*, esperando confirmación por parte del “Empleado”.

Si el *Initiator* no recibe confirmación o se corta la comunicación entre medias, ambos dispositivos pueden iniciar una nueva comunicación *peer to peer*.

En la figura 4.10 se detalla el intercambio de mensajes entre los dos dispositivos.

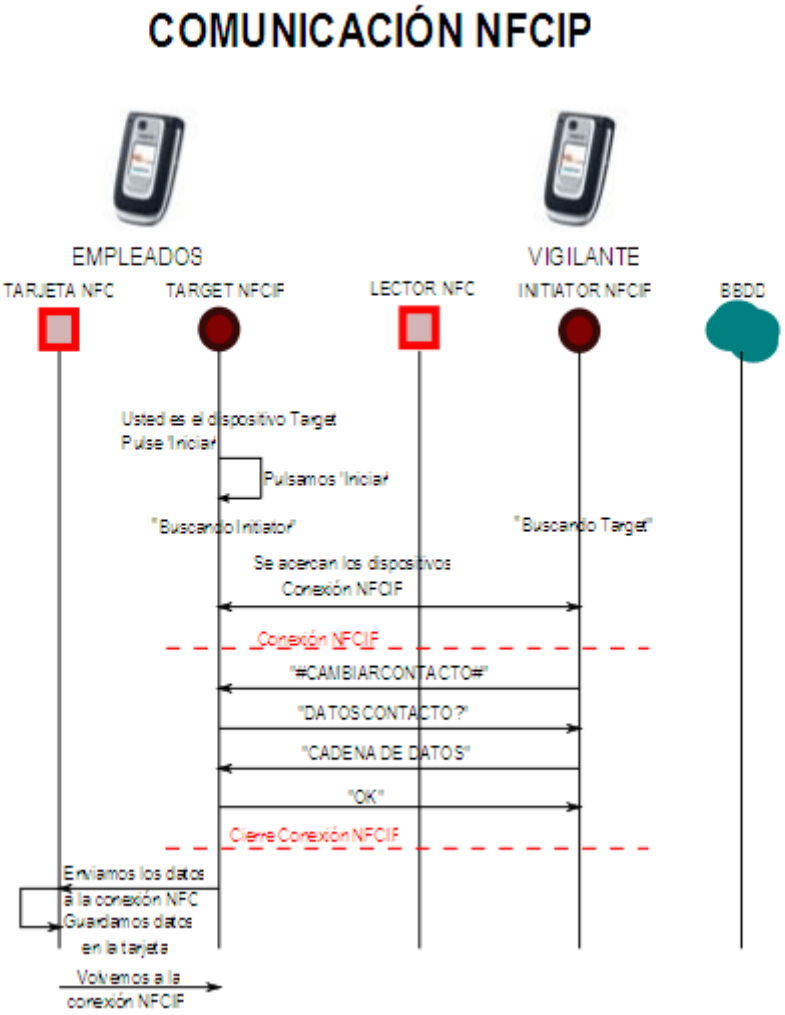


Figura 4.10: Comunicación NFCIP

4.2.2.3. COMUNICACIÓN BLUETOOTH

Esta comunicación se establece entre los módulos Lector Fijo y Vigilante con el módulo Servidor Bluetooth.

Las aplicaciones de los módulos Lector Fijo y Vigilante buscaran únicamente los Servidores que estén definidos en la aplicación, impidiendo que sea posible localizar otros dispositivos que estén conectados con la tecnología Bluetooth.

En el diseño de esta comunicación se decidió que esto fuese así, para reducir el tiempo de búsqueda de dispositivos e impedir que se conectasen a otros dispositivos que no fuesen los Servidores propios de la empresa.

Una vez que ha localizado los servidores, se buscará en ellos, uno por uno, si ofrecen el servicio de búsqueda a la BBDD, definido también previamente. Cuando se encuentra uno que ofrece este servicio establece conexión con él. Únicamente se puede establecer una conexión Bluetooth.

Por otro lado la aplicación del módulo Servidor Bluetooth, es capaz de establecer múltiples conexiones simultáneamente. Crea una conexión nueva por cada petición que recibe de conexión. Así cualquier dispositivo que requiera sus servicios podrá disfrutar de ellos.

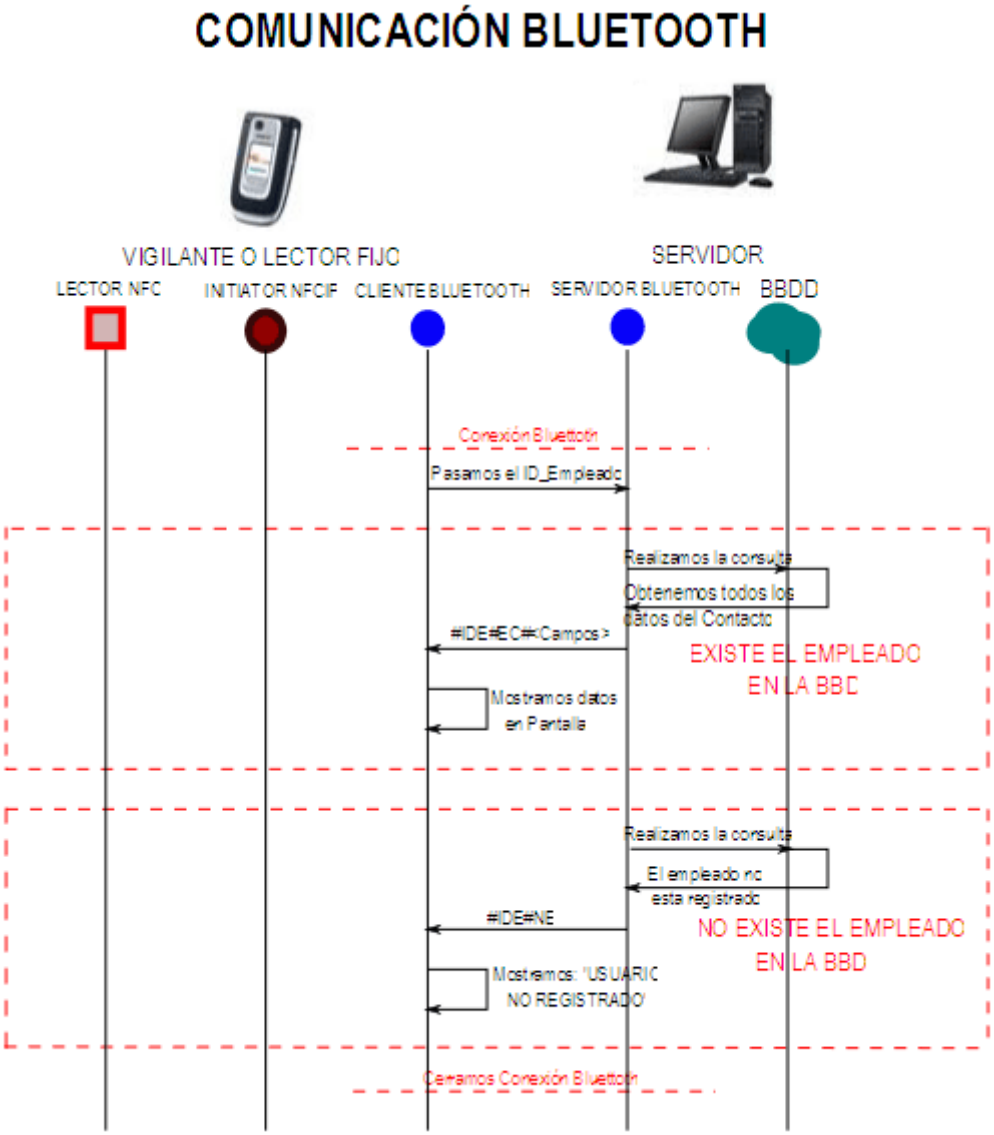


Figura 4.11: Comunicación Bluetooth

4.2.2.4. COMUNICACIÓN CON LA BBDD

Esta comunicación se realiza entre los módulos Servidor Bluetooth y Red Hotel. Esta comunicación se establece entre la aplicación del primer módulo y la BBDD del segundo.

Como se vio en el capítulo 2 en la sección 'Elemento seguro y tarjeta *Mifare* 4k Interna', cada etiqueta *Mifare* contiene en el primer bloque cuatro bytes que definen el UID[25], identificador único.

Se decidió que en la comunicación Bluetooth, al pedir la información sobre un empleado al Servidor, únicamente se enviase este dato, para optimizar recursos.

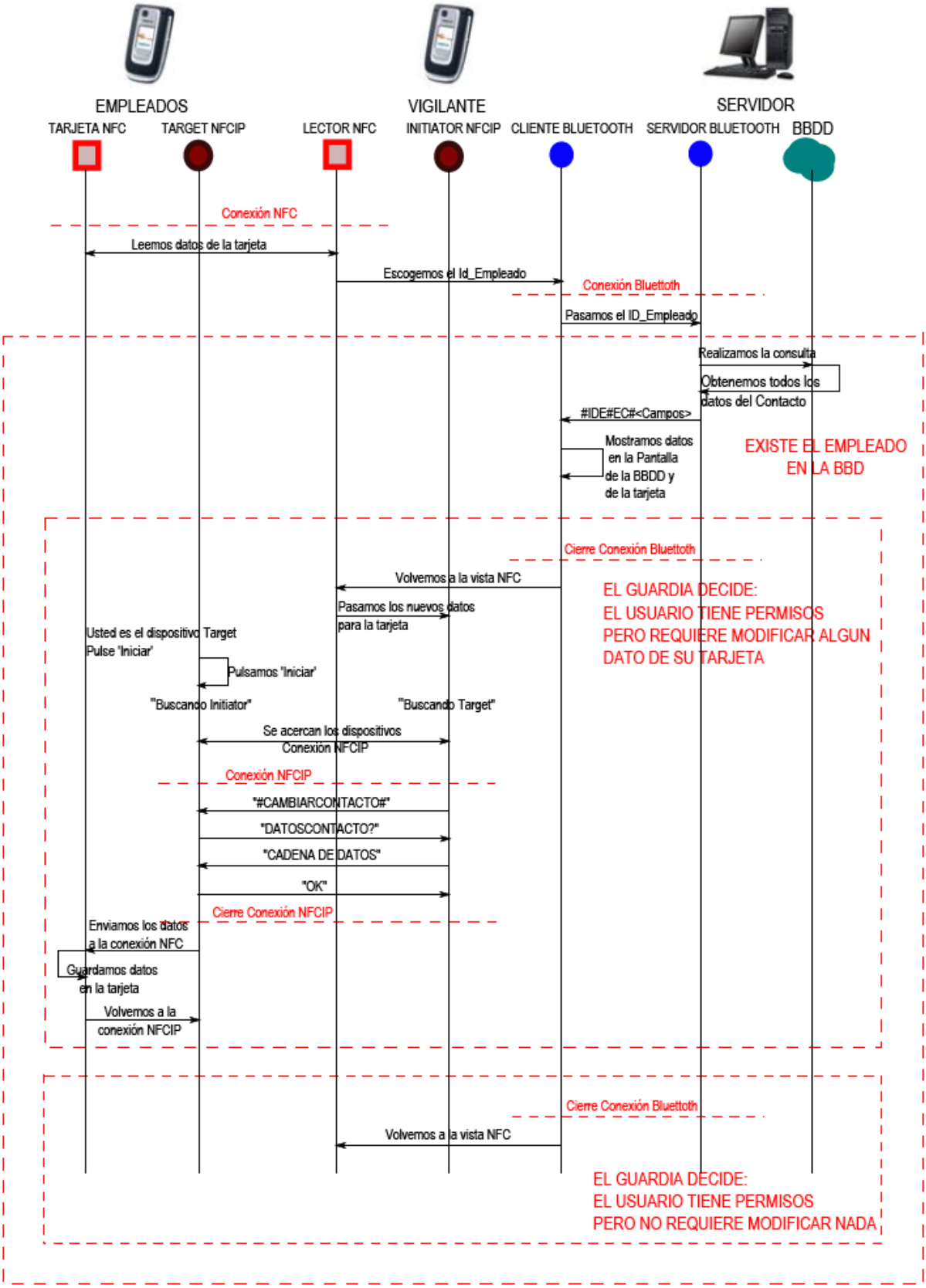
El servidor, una vez recibe este dato, consulta la BBDD donde están todos los empleados y comprueba que hay un empleado registrado con ese Identificador.

4.2.3. ESQUEMA GLOBAL

Una vez hemos definido los diferentes módulos que conforman el proyecto, y su estructura lógica, detallaremos la iteración de cada una de las partes con el resto, formando un sistema de control de acceso y sistemas de identificación.

[25] UID: Unique Identifier

ESTRUCTURA DEL VIGILANTE



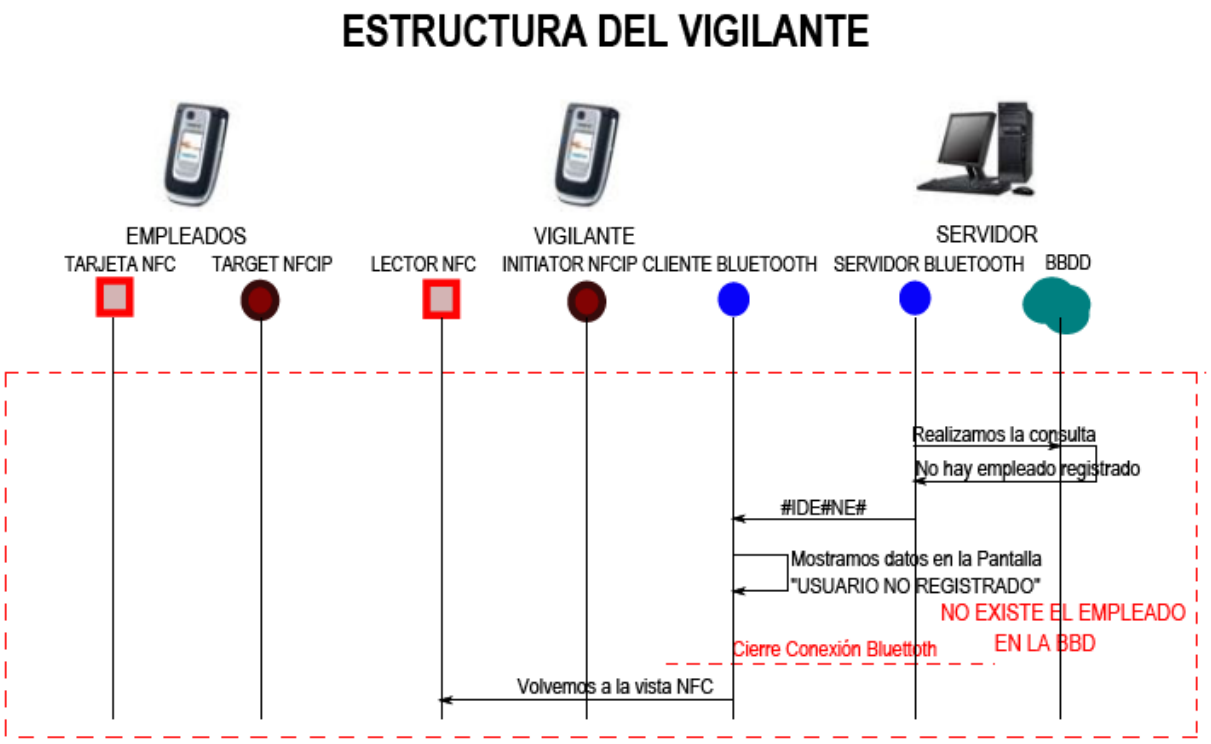


Figura 4.12: Estructura General del Vigilante

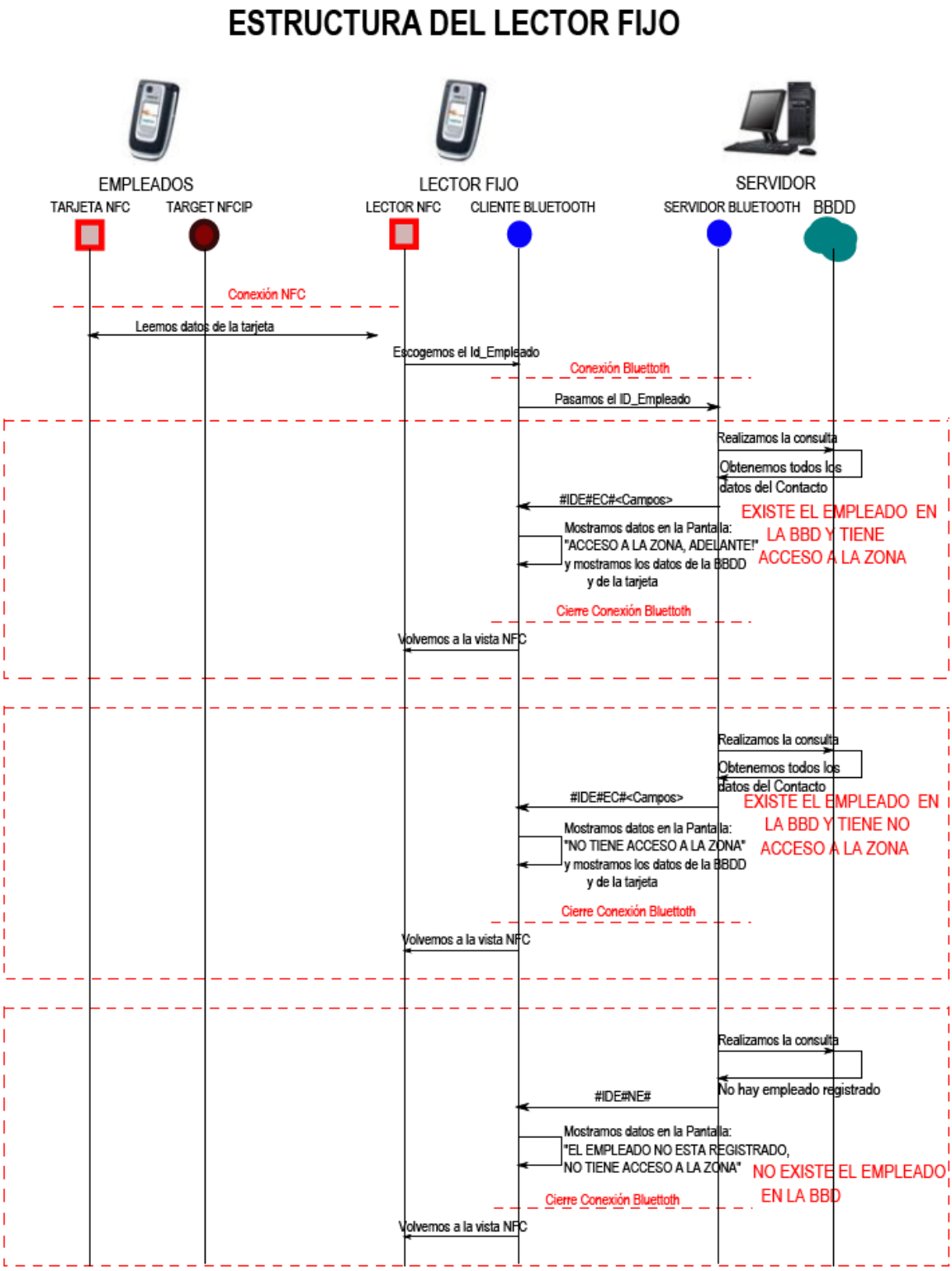


Figura 4.13: Estructura General del Lector Fijo

CAPÍTULO 5

IMPLEMENTACIÓN

En este capítulo se van a detallar todos los pasos que se han seguido para la realización de la aplicación. Una vez realizado el diseño a alto nivel de la aplicación, se puede comenzar con la implementación de todos los componentes.

La dificultad depende de factores como el lenguaje de programación, la metodología empleada y el entorno de desarrollo establecido.

En primer lugar se hará una breve mención al entorno de desarrollo y al lenguaje de programación. A continuación se detalla la implementación de cada una de las comunicaciones definidas en el capitulo anterior y posteriormente se presenta un diagrama de clases para cada una de las aplicaciones.

5.1. LENGUAJE DE PROGRAMACIÓN Y ENTORNO DE DESARROLLO

Las aplicaciones a implementar para el desarrollo de este proyecto están hechas con Java 2 *Micro Edition*. Se ha escogido esta plataforma porque el móvil soporta aplicaciones Java y la comunicación entre la aplicación que corre dentro del móvil y el modulo NFC integrado en él, la realiza mediante la especificación JSR-257.

Concretamente se utilizo CLDC 1.1 (*Connected Limited Device Configuration*), que define la base de interfaces de programación y una maquina virtual para dispositivos de recursos limitados, como el caso del teléfono móvil Nokia 6131 NFC empleado en este proyecto.

Sobre esta configuración, se empleo el perfil MIDP 2.0 (*Mobile Information Device Profile*), que provee una plataforma de desarrollo de aplicaciones Java que correr en equipos con limitada memoria, capacidad de procesamiento y capacidad grafica.

El entorno de desarrollo utilizado para la programación en el móvil es *Netbeans* 9.0, porque nos ofrece las características necesarias para la implementación de la aplicación *MIDlet* en el dispositivo móvil. También nos permite la simulación de la comunicación NFC mediante un *plugin* que integra la Plataforma Series 40 SDK creada por Nokia.

5.2. COMUNICACIONES IMPLEMENTADAS

En este apartado detallaremos la implementación de cada una de las comunicaciones definidas en el Capitulo anterior.

5.2.1. COMUNICACIÓN NFC

En el Capitulo 2 vimos el API para la comunicación NFC. A continuación detallamos los detalles de la implementación para el inicio de la comunicación NFC y para la transmisión de datos mediante esta comunicación.

5.2.1.1. INICIO DE LA COMUNICACIÓN NFC

Para establecer una comunicación NFC en cualquiera de las aplicaciones, el primer paso será importar las librerías que habilitarán esta comunicación. Como se mencionó el paquete principal para la comunicación NFC es *javax.microedition.contactless*.

Se debe implementar la interfaz *TargetListener()* para poder descubrir los objetos soportados. Cuando detecte una tarjeta se genera un evento y podremos crear la conexión. Es necesario registrar el *TargetListener()* en el *DiscoveryManager.*

```
import javax.microedition.contactless.*;
import.javax.microedition.contactless.TargetListener;
      …
      DiscoveryManger dm = DiscoveryManager.getInstance();
      Try{
            …
            dm.addTargetListener(this, TargetType.RFID_TAG);
            …
      }
      Catch(ContactlessException e){
            //Manejar la excepción
      }
```

Por tanto este código nos permite descubrir y registrar las etiquetas que estén al alcance del dispositivo, pero solamente se puede establecer una sola comunicación con una etiqueta. Al utilizar la clase *TargetType()*, le pedimos que busque solamente los tipos de etiquetas soportadas.

Una vez que se ha descubierto la etiqueta, el *DiscoveryManager*, se encargará de llamar al método *TargetDetected(TargetProperties[])*. El array que *TargetPropierties[]*, contiene las tarjetas detectadas, cada una de ellas incluye una URL que representa la conexión con el dispositivo móvil.

```
public void TargetDetected(TargetPropierties[]){
      …
      try{
            …
            //Abrimos conexión con la tarjeta
            MFStandardConnection con = null;
            try{
                  …
                  Con=(MFStandardConnection)Connector.open(tProp[0].get
                  Url("com.nokia.nfc.nxp.mfstd.MFStandardConnection"));
                  …
                  //Aquí se procederá a leer la etiqueta o
                  escribir en ella
            } catch(Exception e){ //Manejamos la excepción }
            …
      }catch(Exception e){//Manejamos la excepción}
      …
      finally {
```

```
            try {
            conn.close();
           } catch ( IOException e ) { //Manejamos la excepción}
    }
}
```

5.2.1.2. ESCRITURA Y LECTURA EN LA COMUNICACIÓN NFC

Una vez se invoca al método *TargetDetected(TargetPropierties[])*, se puede establecer la comunicación con la etiqueta. Ahora es posible realizar transacciones entre el teléfono móvil y la etiqueta, como lectura y escritura.

Como detallamos en el capítulo 2 en la sección 'Estándar de la comunicación' dentro de los estándares de NFC se ha establecido un formato común de datos para que los dispositivos NFC puedan compartir información entre sí, llamado NDEF (*Data Exchange Format)*.

Guardar la información de los empleados en las tarjetas mediante este formato seria lo idóneo. Para ello se necesitaría establecer una comunicación *NDEFConnection.* Pero una de las claves del proyecto era proporcionar seguridad a la información alojada en las tarjetas, por ello se modificaron las claves de los sectores. Esto no es compatible con este tipo de conexión. Esto quiere decir que si modificamos esas claves, la única forma de poder acceder a las tarjetas para guardar información es mediante conexiones *MFStandarConnection*.

Este tipo de conexiones nos permite tener un acceso a bajo nivel, a bloques o sectores.

Escritura de datos

Para la escritura de los datos personales en las tarjetas, estableceremos una conexión *MFStandarConnection* como se especificó en el punto anterior. Se puede escribir directamente en la etiqueta mediante el método *MFStandardConnection.write()*, pero es aconsejable utilizar las clases especiales de la API.

El primer paso es acceder al bloque donde se quieren guardar los datos, y seguidamente haciendo uso de la *KeyA*, escribir los datos.

```
…
try{
      …
      MFBlock block = con.getBlock(pos);
      block.write(keyA, datos, 0);
      …
}
catch(Exception e){
      //Manejamos la excepción
}
```

Lectura de datos

Para la lectura de los datos guardados, tendremos que acceder al bloque donde están guardados, y leer los datos con la *KeyA*:

```
…
byte[] buf = new byte[16];
try{
      …
      MFBlock block = connection.getBlock(pos);
      block.read(keyA, buf,0,0,16);
      …
}
catch(Exception e){
      //Manejamos la excepción
}
```

5.2.2. COMUNICACIÓN NFCIP

En la comunicación *peer to peer* el *MIDlet* puede funcionar siendo *Initiator*, que es el que inicia la comunicación, envía los datos y espera recibir una respuesta, o *Target* que recibe los datos y después envía una respuesta.

Cuando se establece la conexión, la URL especifica que *MIDlet* será el *Initiator* y que *MIDlet* será el *Target*.

```
String INITIATOR_URL = "nfc:rf;type=nfcip;mode=initiator";
String TARGET_URL = "nfc:rf;type=nfcip;mode=target";
```

La conexión se abre de forma habitual:

```
NFCIPConnection con = (NFCIPConnection)Connector.open(url);
```

Target

```
…

Connection con = null;
try{
      …
      con = Connector.open("nfc:rf;type=nfcip;mode=target");
      if(con instanceof NFCIPConnection){
            byte[] receiveBuffer;
            NFCIPConnection connection = (NFCIPConnection)con;
            receiveBuffer = connection.receive();
            String sendBuffer = "datosEnviar";
            connection.send(sendBuffer.getBytes("UTF-8"));
            …
      }
      catch(Exception e){//Manejamos la excepción }
```

El dispositivo "*Target*", espera recibir datos, y cuando los recibe puede enviar datos.

Initiator

```
…
Connection con = null;
try{
      …
      con = Connector.open("nfc:rf;type=nfcip;mode=initiator");
      if(con instanceof NFCIPConnection){
            String sendBuffer = "datosEnviar";
            connection.send(sendBuffer.getBytes("UTF-8"));
            byte[] receiveBuffer;
            NFCIPConnection connection = (NFCIPConnection)con;
            receiveBuffer = connection.receive();
            …
      }
      catch(Exception e){//Manejamos la excepción }
```

El intercambio de mensajes entre ambos dispositivos se detalla en el capítulo anterior.

Al final la conexión deberá ser cerrada:

```
con.close();
```

5.2.3. COMUNICACIÓN BLUETOOTH

En toda comunicación Bluetooth, hay un dispositivo que ofrece un servicio (Servidor) y otro u otros que acceden a ese servicio (Clientes). En nuestro proyecto, el "Vigilante" y el "Lector Fijo" serán los clientes (acceden al servicio de consulta) y el PC será el servidor (realizara las consultas en la BBDD).

Dependiendo de la parte que queramos implementar deberemos tener en cuenta diferentes aspectos:

El Servidor deberá realizar:

- Crear Conexión servidora.
- Ofrecer el servicio.
- Abrir conexiones con los clientes.

Mientras que los Clientes deberán realizar:

- Búsqueda de dispositivos, que se encuentren a su alcance.
- Búsqueda de servicios, que ofrecen esos dispositivos.
- Establecimiento de la conexión, con el dispositivo encontrado, y que ofrece los servicios que buscamos.

A continuación detallamos la implementación de cada parte.

5.2.3.1. SERVIDOR

Crear conexión servidora

Como se definió en el capítulo 3 en la sección 'Bluetooth API JSR-82', la clase *LocalDevice* representa al dispositivo local. Únicamente se debe tener instanciada una sola clase, para ello empleamos el método *getLocalDevice()*. Para definir la visibilidad del dispositivo se emplea el método *setDiscoverable(int mode)*.

El servidor se ha definido con el parámetro *DiscoveryAgent.GIAC()*, que permite tener una conectividad ilimitada.

```
…
LocalDevice dispositivoLocal = LocalDevice.getLocalDevice();
dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
…
```

Ofrecer un servicio

Para utilizar un servicio del Servidor, el Cliente debe utilizar el mismo protocolo que éste. Entre los protocolos disponibles, hemos utilizado el protocolo RFCOMM, que proporciona emulación de múltiples puertos serie RS-232 entre los dispositivos.

La aplicación Servidor ofrece un servicio basado en el Perfil de Puerto Serie (*Serial Port Profile SPP*), por tanto será un servidor SPP. La URL de conexión para el Servidor será de la siguiente forma:

```
…
btspp://localhost:UUID;[parametrosdelservidor]
…
```

Cada servicio se identifica numéricamente con una secuencia de 1 a 32 números en hexadecimal, de tal manera que una vez asignado para referirnos al servicio lo haremos a través de este número asociado. Se denomina UUID (*Universal Unique IDentifier*). El parámetro del servicio que utilizamos para nuestro proyecto es *name*.

Abrir conexiones con los clientes

Una vez definida la URL del servidor, únicamente tendremos que crear la conexión servidora, para ello utilizamos un objeto *StreamConnectionNotifier*. El método *acceptAndOpen()* nos permite indicar que el Servidor está listo para aceptar conexiones con el Cliente, bloqueando al Servidor hasta que algún Cliente se conecta.

```
…
StreamConnectionNotifier servidor =
(StreamConnectionNotifier)Connector.open(URL);
StreamConnection sc = servidor.acceptAndOpen();
…
```

El servidor puede aceptar múltiples conexiones de diferentes clientes llamando a *acceptAndOpen()* varias veces. Se creara un nuevo objeto *StreamConnection* por cada conexión aceptada. El Cliente accede al mismo registro de servicio y conecta el servicio usando el mismo canal RFCOMM del servidor.

Una vez aceptada una conexión con el Cliente, obtenemos el objeto remoto (Cliente), con el que queremos conectarnos, para poder comunicarnos con él, y abrimos los flujos de escritura y lectura, *DataInputStream* y *DataOutputStream* (devuelven flujos de lectura y escritura de bytes, orientados al envió y recepción de texto).

```
…
RemoteDevice rd = null;
rd = rd.getRemoteDevice(sc);
DataInputStream in = sc.openDataInputStream();
DataOutputStream out = sc.openDataOutputStream();
…
```

5.2.3.2. CLIENTE

Búsqueda de dispositivos

Como en la parte del Servidor el primer paso será tener un objeto de la clase *LocalDevice*, que representará al dispositivo en el que se está ejecutando la aplicación.

```
…
LocalDevice dispositivoLocal = LocalDevice.getLocalDevicce();
dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
DiscoveryAgent da = dispositivoLocal.getDiscoveryAgent();
da.startInquiry(DiscoveryAgent.GIAC, new Listener());
…
```

Como vemos utilizamos la clase *DiscoveryAgent*, que nos proporciona un método para descubrimiento de servicios y dispositivos.

Para los segundos están los métodos *startInquiry()* (no bloqueante) o *retrieveDevices()* (bloqueante). El primer método requiere que la aplicación especifique un *Listener*. Éste es avisado cuando encuentra nuevos dispositivos. Si ha hecho una búsqueda previa de los dispositivos y no requiere realizar la búsqueda, se utilizará el segundo método, que devuelve una lista con los dispositivos encontrados en búsquedas previas.

La interfaz *DiscoveryListener()*, permite definir un *Listener* de eventos que responda cuando se encuentren dispositivos o servicios. Tiene dos métodos para búsqueda de dispositivos:

- *deviceDiscovered()*, que se llama cada vez que se encuentra un dispositivo. En nuestra aplicación lo que hará será guardar en un *Vector()*, todos los dispositivos que encuentre.

```
…
Vector dispositivos_encontrados = new Vector();
public void deviceDiscovered(RemoteDevice dispositivoRemoto,
DeviceClass clase){
      …
      Dispositivos_encontrados.addElement(dispositivoRemoto);
      …
}
```

- *inquiryComplete()*, que se llamada cuando la búsqueda a finalizado. En nuestra aplicación, de todos los dispositivos encontrados, seleccionamos el primero que se encuentre definido en nuestra aplicación (Servidor propio de la empresa).

```
public void inquiryCompleted(int completado){
      …
      for(int i=0; i<dispositivos_encontrados.size(); i++){
            if(obtenerNombreDispositivo(dispositivos_encontrados.elemen
            tAt(i).equalsIgnoreCase("nombre de los servidores del
            Hotel"))){
                  startServiceSearch(RemoteDevice)dispositivos_encontra
                  dos.elemetAt(i);
            }
      }
      …
}
```

Búsqueda de servicios

Una vez que se ha realizado la búsqueda de dispositivos, el siguiente paso será encontrar los servicios que ofrece. Al igual que con los dispositivos, la búsqueda de los servicio se realiza también con el objeto *DiscoveryAgent*. Utilizando el método *searchServicer()*, al que se le pasa de nuevo un objeto *DiscoveryListener()*, el dispositivo sobre el que queremos realizar la búsqueda (device), y los servicios en los que estamos interesados (uuids).

```
…
da.searchServices(null, uuids, device, new Listener());
…
```

Cuando se encuentra al algún servicio se notifica a la aplicación a través del *Listener*, con el método *serviceDiscovered()*. Este método recibe un array de objetos *ServiceRecord* que contienen los atributos de servicio que se han solicitado. Obtenemos para nuestra aplicación la URL necesaria para establece conexión con el Servidor.

```
…
String url =
rec.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
enviarmensaje("Mensaje", url);
…
```

Como vemos, después de haber obtenido la url, llamaremos al método "enviarMensaje", que será el encargado de abrir conexión con el Servidor.

Establecimiento de la conexión

Una vez que tenemos la URL del Servidor que nos ofrece el servicio buscado, tendremos que abrir una conexión con él. Para ello utilizamos un objeto *StreamConnectionNotifier.* Una vez establecida la conexión con el Servidor, abriremos los flujos de lectura y escritura *DataInputStream* y *DataOutputStream* para comunicarnos con él.

5.2.4. COMUNICACIÓN CON LA BBDD

Para la comunicación con la BBDD utilizamos, JDBC, una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea.

La aplicación de Java debe tener acceso a un controlador (*driver*) JDBC adecuado. Este controlador es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real. De manera muy simple, al usar JDBC se pueden hacer tres cosas:

- Establecer una conexión a la BBDD.

```
…
try{
      Connection con =
DriveManager.getConnection("jdbc:mysql://localhost…..);
} catch(SQLException e){…}
…
```

- Mandar consultas y sentencias a la fuente de datos.

```
…
String consulta = "SELECT apellido, nombre FROM ……";
ResultSet rs = st.executeQuery(consulta);
…
```

- Procesar los resultados.

```
…
While(rs.next()){
//Se procesara la consulta
}
…
```

Toda la conectividad de bases de datos de Java se basa en sentencias SQL, por lo que se hace imprescindible un conocimiento adecuado de SQL para realizar cualquier clase de operación de bases de datos.

5.3. DIAGRAMA DE CLASES

Una vez detallados los aspectos más importantes del proyecto, vemos los diagramas de clase de cada una de las aplicaciones, y su implementación más en detalle.

5.3.1. APLICACIONES ADICIONALES

5.3.1.1. APLICACIÓN SECUREWRITERMF

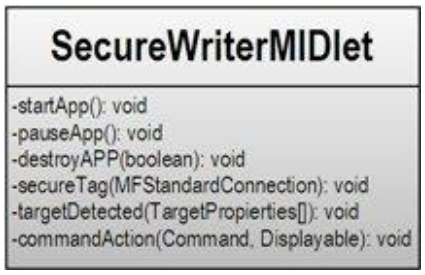


Figura 5.1: Diagrama de Clase *SecureWriterMF*

Como vimos en la parte de diseño, esta aplicación es un simple *MIDlet* J2ME que cambia el valor del contenido de los sectores de remolque de los sectores donde se desee escribir.

El dispositivo móvil funcionara en modo pasivo, siendo él, el que genera un campo magnético permitiendo la transferencia de información a la tarjeta. Para la implementación de funcionalidades NFC en J2ME, se necesita hacer uso del API JSR-257.

Para nuestra aplicación haremos uso del paquete *javax.microedition.contactless* para el descubrimiento de las tarjetas y establecer las conexiones necesarias, y el paquete adicional *com.nokia.nfc.nxp.mfstd* para la transmisión de datos, cambio de claves.

El método que nos permite realizar el cambio de las claves es *writerTrailerContents(MFKey, MFTrailerContents);* donde *MFKey* es la clave que hay actualmente y *MFTrailerContents*, son los bits del sector de remolque con las nuevas claves. Como vemos para modificar las claves es necesario sobrescribir el bloque entero donde están contenidas las llaves (bloque de remolque).

Cuando iniciamos el *MIDlet*, únicamente debemos acercar la tarjeta para el cambio de claves. Para ello inicializaremos la comunicación NFC como se ha explicado anteriormente. Cuando se abre la conexión *MFStandarConnection*, se hace una llamada al método *SecureTag()*, que es el encargado de realizar el cambio.

5.3.1.2. APLICACIÓN ACCESSSECURETAG

Esta aplicación tiene cuatro partes, dos de ellas para la lectura y escritura del elemento seguro, las clases Read y Write. Y dos de ellas para la simulación del elemento seguro, lectura y escritura en tarjetas externas, las clases ReadExternal y WriteExternal.

Cuando inicializamos el *MIDlet* nos saldrán las cuatro opciones a escoger. “Leer” y “Escribir” para el elemento seguro del dispositivo, que solo podremos utilizar en el simulador de Nokia. Y “Leer Externo” y “Escribir Externo” que utilizaremos en nuestro proyecto para leer y escribir los datos en las tarjetas externas.

Las clases con las que cuenta esta aplicación son:

- Clase AccessSecureTagMIDlet:

Es la clase principal de esta aplicación. Extiende de *Midlet*, por lo tanto arranca la aplicación, además implementa *CommandListener*, pudiendo acceder a las distintas opciones que proporciona. Los métodos que tiene son los propios de acuerdo a estas dos características, *startApp()*, *pauseApp()*, *destroyApp()*, *commandAction()*, además contamos con:

-*getDisplay()*: Obtiene el *display* que está lanzado.

-*setDisplay()*: Modifica el *display* que se va a lanzar.

- Clase ReadExternal:

Muestra por pantalla un mensaje donde se solicita que se acerque la tarjeta que se desea leer. Extiende de *Form*, por lo que contamos con el método *init()*, que obtenemos el lector NFC si es *null*.

-*ReadExternal()*: Constructor que inicializa las variables.

-*clearForm()*: Limpia la pantalla del *Form*.

-*getMyTagReader()*: Devuelve el lector NFC.

-*obtenerDato(String d)*: Nos permite obtener el valor de cada dato a partir de una cadena.

-*mostrarContacto()*: Muestra por pantalla los datos leídos del empleado.

- Clase Read:

Esta clase tiene la misma funcionalidad que la anterior, pero en vez de leer etiquetas externas, se empleará para la lectura de la etiqueta interna. La diferencia es que esta clase implementa *CommandListener*, por lo que en el método *init()*, inicializaremos los botones necesarios. Además de los métodos de la clase anterior contamos con *commandAction()*, que gestionara los eventos de los botones.

- Clase WriteExternal:

Esta clase muestra los datos del empleado que se van a guardar en la tarjeta externa. Extiende de *Form*, por lo que también contara con el método *init()*, que nos permitirá escribir los datos a guardar, además de obtener el escritor NFC si es *null*.

-*WriteExternal()*: Constructor que inicializa las variables.

-*clearForm()*: limpia la pantalla del *Form*.

-*getWriterContact()*: Devuelve el escritor NFC.

-*escribirTarjeta()*: Avisa si no se ha introducido ningún dato. Y llama al escrito NFC para guardar los datos.

- Clase Write:

Tiene la misma funcionalidad que la anterior, pero para la etiqueta interna del dispositivo. A diferencia de este, implementa *CommandListener*, por lo que contamos además con el método *commandAction()*, que gestionara las distintas opciones según los botones pulsados.

- Clase MFReader:

Gestiona las operaciones de lectura de la tarjeta, tanto interna como externa. Extiende de *NFCHandler* por lo que aparecen los métodos de esta interfaz.

-*InternalDetectionMade()*: Establecerá conexión con la etiqueta interna del dispositivo, para la lectura de los datos.

-*targetDetectionMade()*:Detecta las propiedades de la tarjeta externa y realiza la conexión con ella, para poder realizar la lectura.

-*getFormattedName()*: Devuelve los datos del empleado guardados en la tarjeta.

-*setFormattedName()*: Este método solo se utiliza para la escritura de datos.

- Clase MFWriter:

Permite escribir en la tarjeta seleccionada, externa o interna. Extiende de *NFCHandler* por lo que tendrá los mismos métodos que la clase anterior. Pero cuando detecta una tarjeta, se establece la conexión para realizar la escritura de datos en vez de para la lectura.

- Clase MFUtil:

Realiza las operaciones de lectura y escritura en los diferentes tipos de tarjetas. Implementa *TargetListener*, por lo que cuenta con el método *targetDetected(TargetPropierties[] t)* que es llamado cuando se detecta una tarjeta, y obtiene sus propiedades. Además de este método, tenemos:

-*MFUtil()*: Constructor que inicializa las variables.

- *setHandler()*: Modifica el manejador de la tarjeta.

-*getMFTAGConnection()*: Gestiona las propiedades de la tarjeta externa detectada y establece conexión con ella.

-*getMFTAGInternalConnection()*: Establece la conexión con la etiqueta interna.

-*writeMFInternalMessage()*: Escribe en la etiqueta interna del dispositivo.

-*writeExternalMessage()*: Escribe en la etiqueta externa.

-*readMFInternalMessage()*: Lee la etiqueta interna del dispositivo.

-*readExternalMessage()*: Lee la etiqueta externa.

-*getUtil()*: Tipo de tarjeta que detecta.

-*obtenerCampo()*: Método que sirve para obtener el campo que se quiere guardar.

-*leerCampo()*: Método que lee datos de la etiqueta en la posición que se le indica.

-*cerrarConexion()*: Método que cierra la conexión *MFStandardConnection* establecida con la tarjeta.

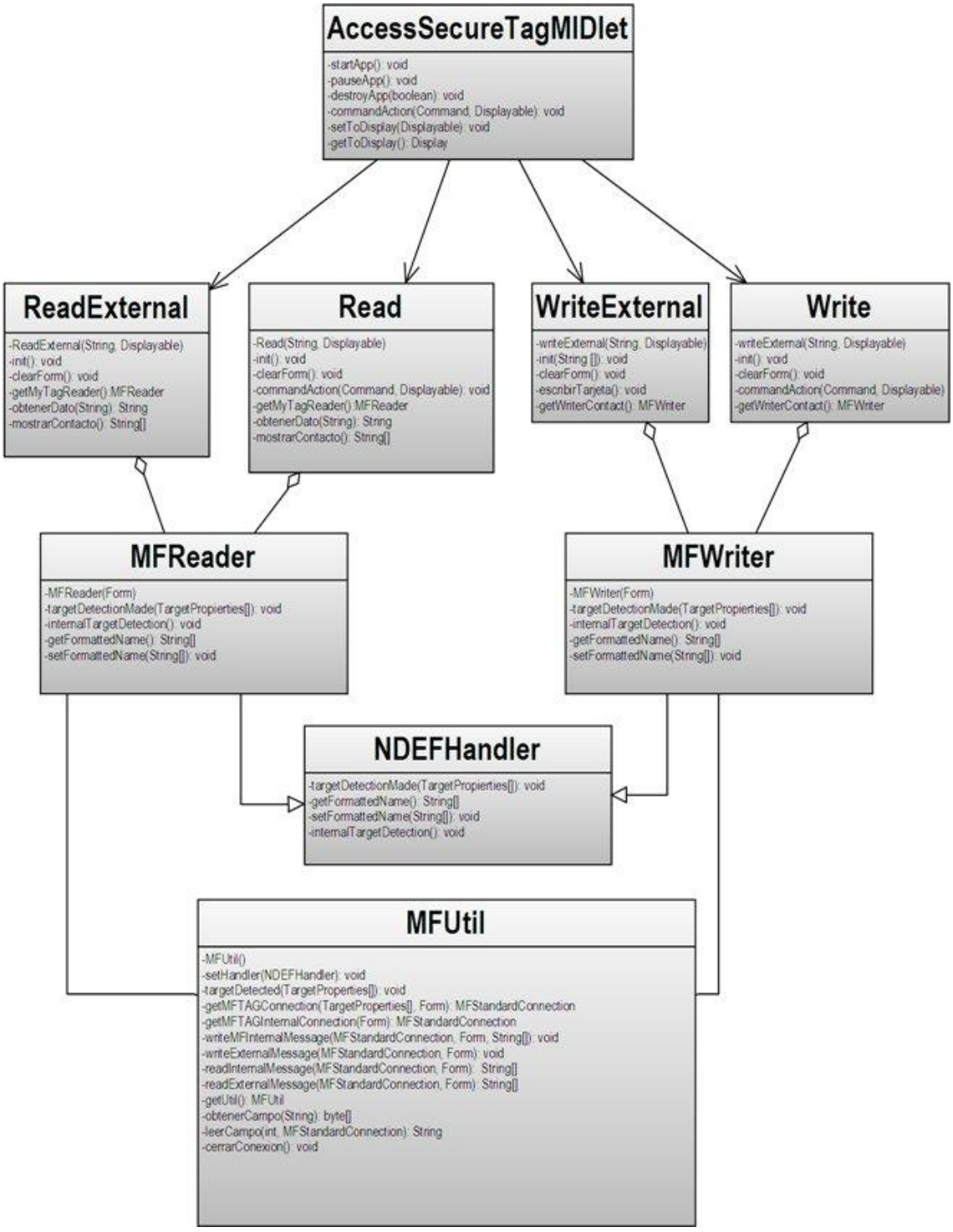


Figura 5.2.: Diagrama de Clases *AccessSecureTag*

5.3.2. MÓDULO EMPLEADO

- Clase EmpleadoMIDlet:

Es la clase principal de esta aplicación. Extiende de *MIDlet*, por lo tanto arranca la aplicación, Implementa *CommandListener*, por lo que contará con el método *commandAction(),* que permitirá acceder a las distintas opciones que proporciona. Los métodos que tiene son los propios de acuerdo a estas dos características, startApp(), pauseApp(), destroyApp(), contando también con:

-*getDisplay()*: Obtiene el *display* que está lanzado.

-*setDisplay()*: Modifica el *display* que se va a lanzar.

-*send()*: Inicializa un Hilo para la comunicación NFCIP. (*Target*)

- Clase Target:

Esta clase gestiona la comunicación NFCIP con el dispositivo VIGILANTE. Implementa *Runnable*, por lo que cuenta con el método *run(),* que se encarga del intercambio de información con el otro dispositivo. Además contamos con:

-*Target()*: constructor que inicializa las variables.

-*obtenerContacto()*: obtenemos los diferentes datos de cada empleado (nombre, apellido o teléfono) a partir de una cadena de texto con todos estos datos.

-*limpiarPantalla()*: Limpiamos la pantalla del *display*.

Además de estas dos clases, la aplicación EMPLEADO cuenta con el modulo completo de la aplicación AccessSecureTag. Todo esto se detalla en el siguiente diagrama de clases.

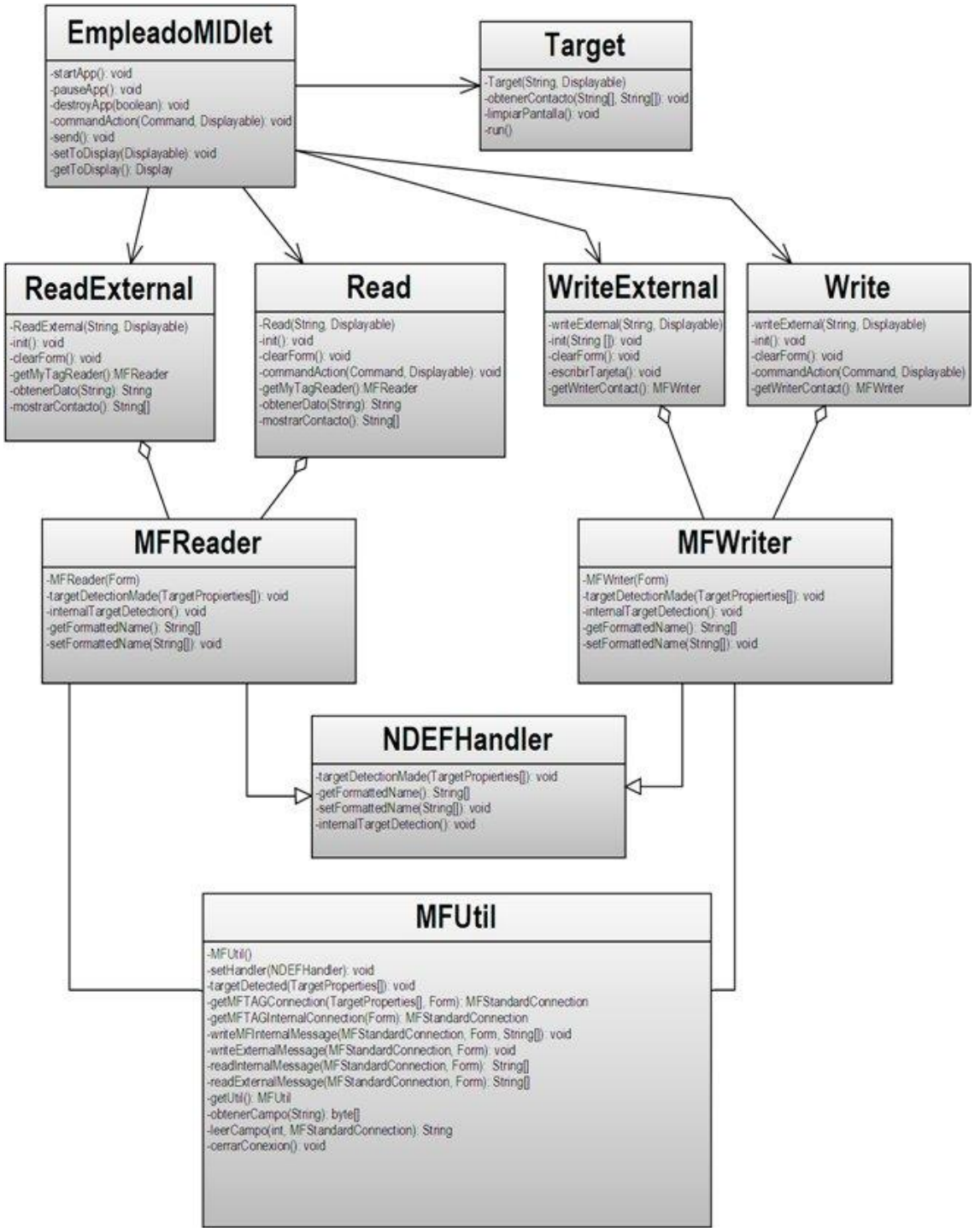


Figura 5.3.: Diagrama de Clases *Empleado*

5.3.3. MÓDULO VIGILANTE

- Clase VigilanteMIDlet:

Es la clase principal de esta aplicación. Extiende de *Midlet*, por lo tanto arranca la aplicación, además implementa *CommandListener*, pudiendo acceder a las distintas opciones que proporciona. Los métodos que tiene son los propios de acuerdo a estas dos características, startApp(), pauseApp(), destroyApp(), commandAction(), además contamos con:

-*send()*: Inicializa un Hilo para la comunicación NFCIP. (*Initiator*).

-*setToDisplay()*: Modifica el *display* que se va a lanzar.

-*getToDisplay()*: Obtiene el *display* que está lanzado.

- Clase Initiator:

Esta clase gestiona la comunicación NFCIP con el dispositivo EMPLEADO. Implementa *Runnable*, por lo que cuenta con el método *run(),* que se encarga del intercambio de información con el otro dispositivo. Además contamos con:

-*Initiator()*: constructor que inicializa las variables.

-*limpiarPantalla()*: Limpiamos la pantalla del *display*.

-*setContacto()*: Guardamos los datos de la tarjeta y de la BBDD.

-*mostrarPantalla()*: Muestra por pantalla los datos que hay guardados en la tarjeta y en la BBDD.

-*escribirPantalla()*: Escribimos un mensaje en la pantalla.

- Clase ClienteBluetooth:

Es la encargada de la comunicación con el servidor Bluetooth. Se compone de:

-*ClienteBluetooth()*: Es el constructor de la clase, encargado de inicializar todas las variables necesarias.

-*salir()*: Pone a *null* todas las variables, y volvemos al anterior *display*.

-*mostrarAlarma()*: muestra mensajes de error con la conexión Bluetooth.

-*cancelar()*: Cancela la búsqueda de servicios en los dispositivos encontrados.

-*desconectar()*: cerramos todas las conexiones abiertas.

-*enviarMensaje()*: Nos comunicamos con el servidor Bluetooth, enviando y recibiendo datos.

-*getContactoBbdd()*: Obtenemos los datos del empleado que tiene almacenados la BBDD.

-*compararDatos()*: Mostramos por pantalla, los datos de la tarjeta y los datos de la BBDD.

-*obtenerDato()*: Obtenemos los datos de la cadena de texto que nos envía el servidor.

-*startDeviceInquiry()*: Creamos un nuevo *Listener* para la conexión Bluetooth.

-*startServiceSearch()*: Comienza la búsqueda de servicios si tiene una lista de dispositivos encontrados anteriormente, creando un nuevo *Listener*.

-*obtenerNombreDispositivo()*: Obtiene el nombre del dispositivo remoto.

- Clase Listener:

Esta clase implementa la interfaz *DiscoveryListener*, y por ello se implementan sus cuatro métodos, *devideDiscovered(), inquiryCompleted(), servicesDiscovered(), serviceSearchComplete()*.

El resto de Clases que aparecen en esta aplicación tienen la misma funcionalidad que en la aplicación Empleado descrita anteriormente. El cambio que encontramos es que aparece en la clase ReadExternal, el método *getUID()*, con el que obtenemos el identificador único de la tarjeta que queremos leer. Del mismo modo, las clases MFReader y MFUtil también lo tienen.

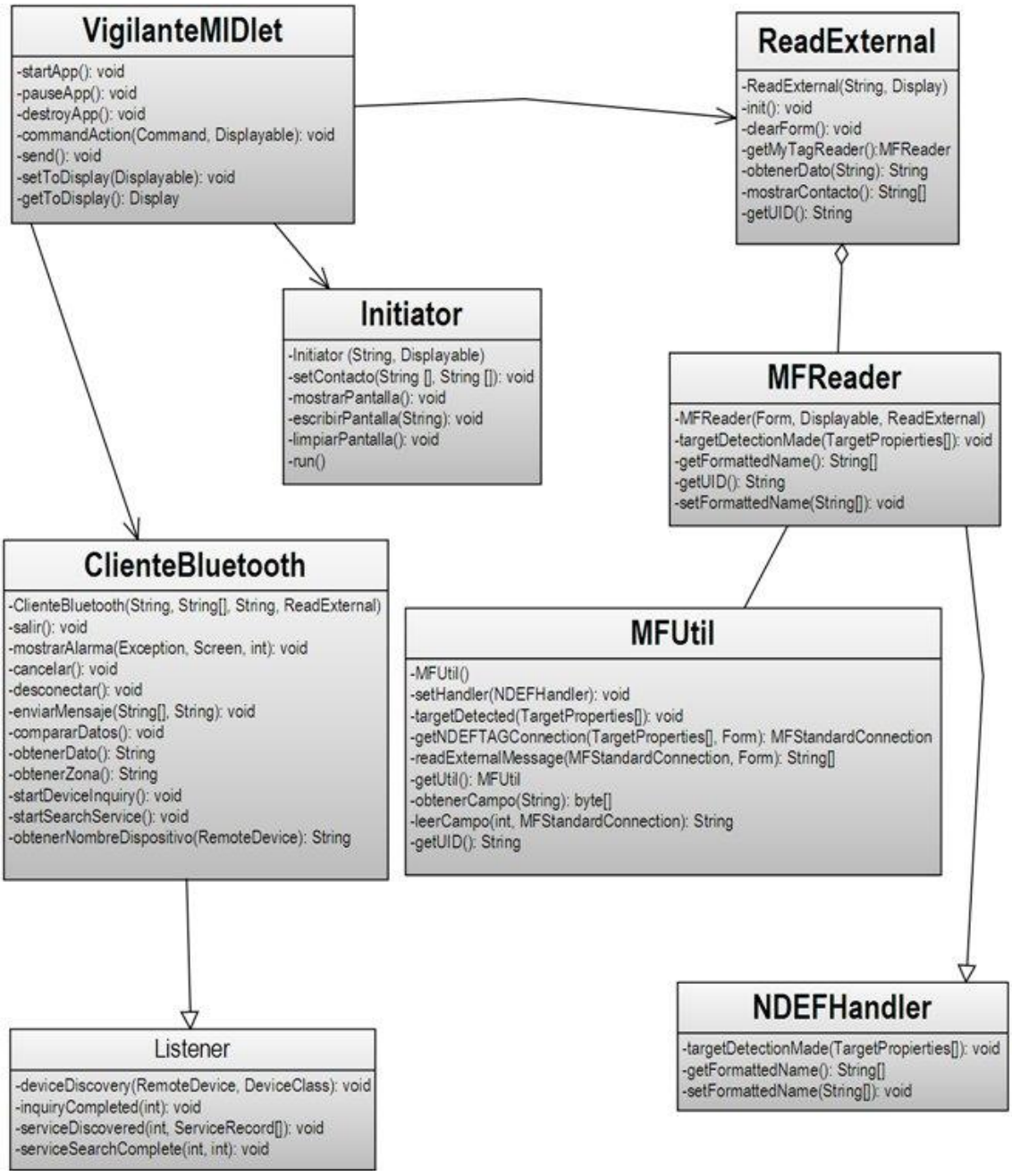


Figura 5.4.: Diagrama de Clases *Vigilante*

5.3.4. MÓDULO LECTOR FIJO

- Clase LectorMIDlet:

Es la clase principal de esta aplicación. Extiende de *Midlet*, por lo tanto arranca la aplicación, además implementa *CommandListener*, pudiendo acceder a las distintas opciones que proporciona. Los métodos que tiene son los propios de acuerdo a estas dos características, startApp(), pauseApp(), destroyApp(), commandAction(), además contamos con:

-*setToDisplay()*: Modifica el *display* que se va a lanzar.

-*getToDisplay()*: Obtiene el *display* que está lanzado.

Las demás clases están definidas en las aplicaciones anteriores, teniendo la misma funcionalidad que estas, a excepción de la clase MFReader que se encargar de comunicarse mediante Bluetooth con el servidor Bluetooth cada vez que se detecte una tarjeta, por lo que el método *targetDetectionMade()* será el encargado de llamar a la clase ClienteBluetooth.

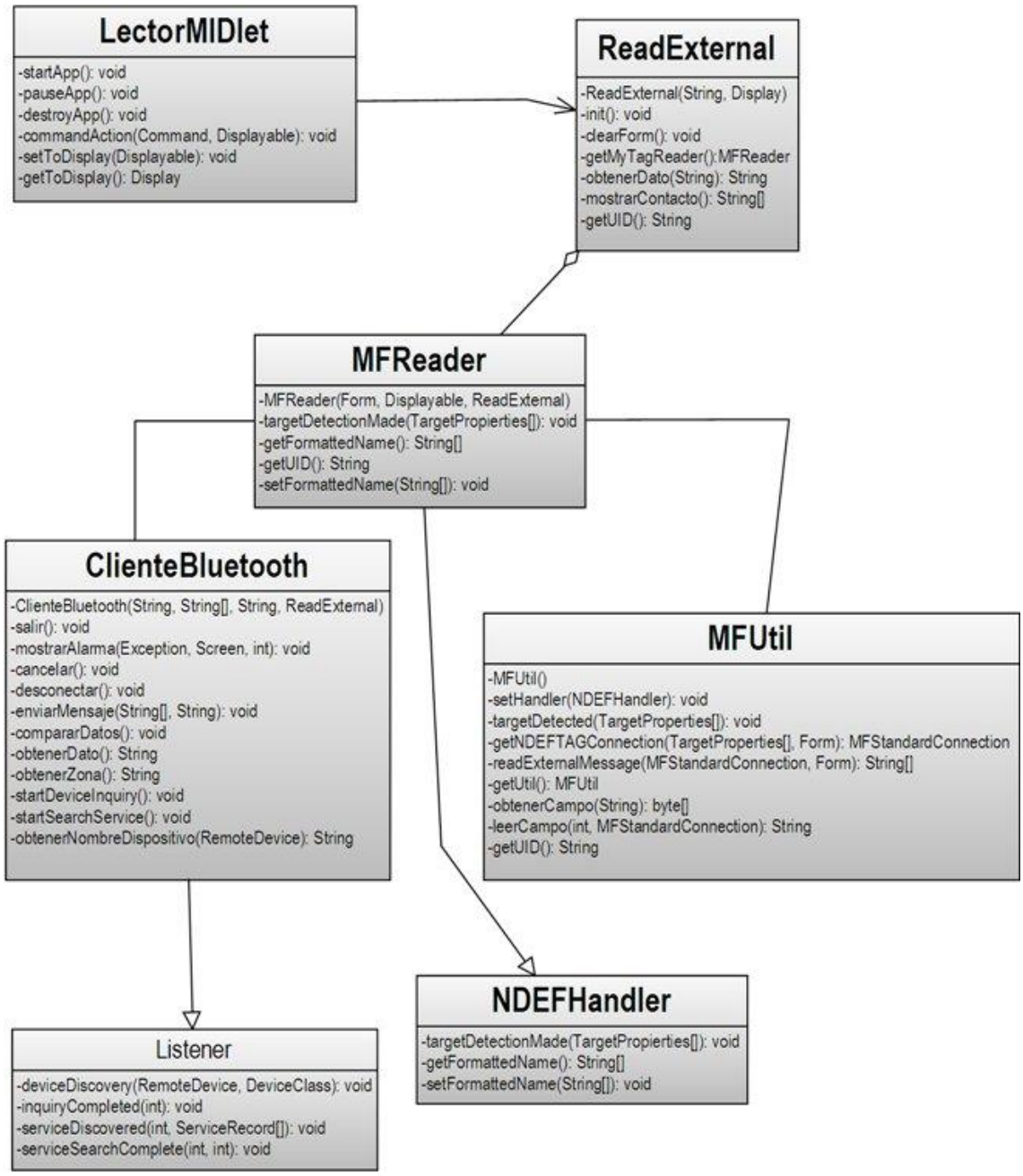


Figura 5.5.: Diagrama de Clases *Lector Fijo*

5.3.5. MÓDULO SERVIDOR BLUETOOTH

- Clase SPPServidor:

Clase principal de esta aplicación. Contiene el método *main()*, que se ejecuta al invocar el programa. Este método crea una instancia de la clase. Además contamos con los métodos:

-*SPPServidor()*: Constructor que inicializa todas las variables necesarias. Además inicializamos la comunicación Bluetooth.

-*mostrarMensaje()*: Muestra por pantalla un mensaje.

-*startThread()*: Este método espera recibir peticiones de conexión Bluetooth. Por cada petición recibida inicializa un hilo para establece la conexión con ese dispositivo.

-*desconectar()*: Cierra todas las conexiones abiertas.

- Clase hiloServidor:

Esta clase implementa *Runnable*, y se inicializa por cada conexión que se desea establecer. Contamos con el método *run()*, que gestiona la comunicación con el dispositivo con el que nos conectamos.

-*hiloServidor()*: Constructor que inicializa las variables necesarias.

-*startWrite()*: Método que envía datos al dispositivo al que está conectado.

-*consultarBasedeDatos()*: Método que obtiene los datos de un empleado.

-*mostrarMensaje()*: Método que muestra por pantalla un mensaje.

-*desconectar()*: Cierra todas las conexiones abiertas, cuando termina la comunicación.

- Clase GestionBBDD:

Esta clase es la encargada de conectarse con la BBDD y obtener los datos que se precisen. El constructor de la clase, será el encargado de establecer la conexión con la BBDD.

-*obtenerNombreUsuarios()*: Obtiene los nombres de los empleados registrados en la BBDD.

-*obtenerUsuarios()*: Obtiene todos los datos de todos los empleados registrados en la BBDD.

-*getZonasAcceso()*: Obtiene las zonas a las que un empleado tiene acceso.

-*getIdHistorial()*: Obtenemos el IdHistorial del día actual.

-*getEntrada()*: Obtenemos la hora de entrada a una zona.

-*setEntrada()*: Introducimos la hora de entrada de un empleado en una zona.

-*setHoraEntrada()*: Introducimos la hora de entrada de un empleado en una zona, al historial correspondiente.

-*setHoraSalida()*: Introducimos la hora de salida de un empleado en una zona, al historial correspondiente.

-*borrardeZona()*: Borra la hora de entrada, cuando el empleado sale de la zona, de una tabla auxiliar.

-*setHistoAbierto()*: Nos dice si tenemos un historial abierto para un empleado, y si no es así, se crea uno.

-*obtenerDato()*: Método auxiliar que nos permite obtener datos cuando están a *null* en la BBDD.

-*obtenerFecha()*: Método auxiliar que nos permite obtener una fecha cuando está a *null* en la BBDD.

-*obtenerNumero()*: Método auxiliar que nos permite obtener una número cuando está a *null* en la BBDD.

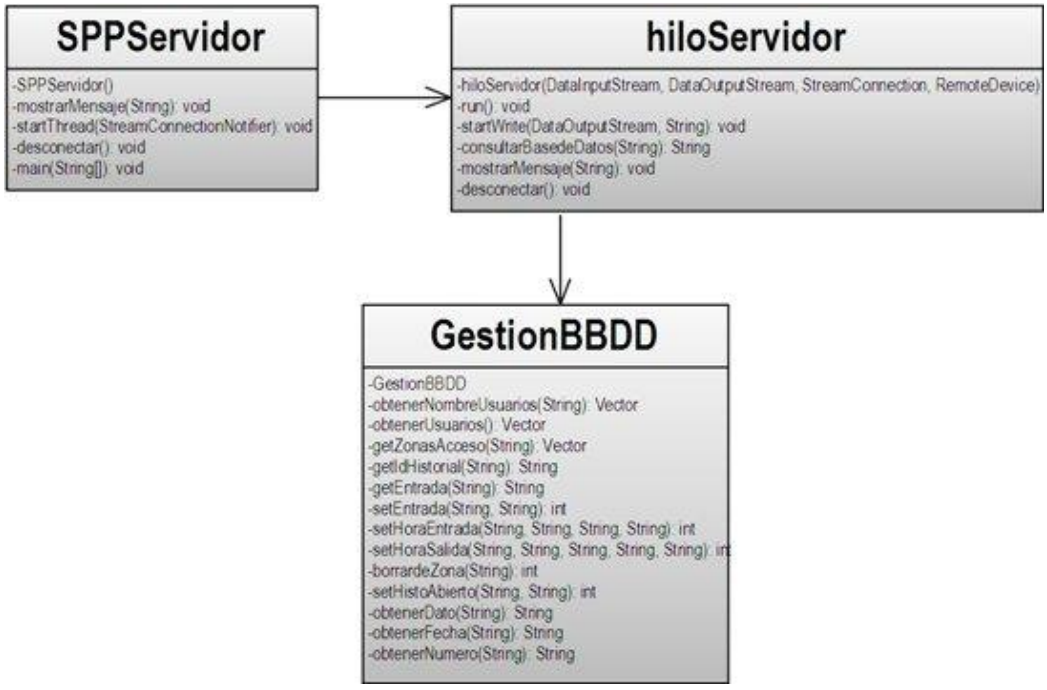


Figura 5.6.: Diagrama de Clases *Servidor Bluetooth*

5.3.6. MÓDULO RED HOTEL

5.3.6.1. APLICACIÓN WEB GESTION HOTEL

Dentro del **paquete Web**, encontramos:

- META-INF: Contiene el archivo de configuración context.xml
- WEB_INF: Contiene el archivo de configuración web.xml y una carpeta donde se encuentran los archivos compilados (*Servlets, Beans*).
- Img, carpeta que contiene las imágenes necesarias.
- Style: Contiene los archivos css.

Además de todo esto encontramos las clases utilizadas por la aplicación web. Los *JSPs*.

En nuestra aplicación web, estos *JSPs* definen varias partes bien diferenciadas:

- *Login*: Para acceder al resto de funcionalidad de la página web, el empleado debe logearse primero, con su nombre y *password*.
- *Empleados*: Una vez que esta logeado, puede acceder a una zona, en la que aparece una tabla con todos los empleados logeados.
- *Modificar Empleado*: Además podrá seleccionar un empleado, y modificar sus datos personales, borrarle o modificar sus zonas de acceso.
- *Zonas*: Contara también con la posibilidad de poder ver las zonas a las que cada empleado puede acceder. Para ello contara con una tabla en la que aparecerán todos los empleados, y las zonas de acceso. Pudiendo modificarse si se desea.
- *Añadir Empleado*: Podemos añadir nuevos empleados, introduciendo sus datos personales y las zonas de acceso.
- *Historial*: Contaremos con otra zona, para poder ver los diferentes historiales de los vigilantes, y tener un control sobre las rutas echas.

Dentro del **paquete Source** encontramos los *Beans* y *Servlets*, necesarios para la consultas con la base de datos.

Los *Beans* son: Persona.java, Usuario.java y Zona.java.

Los *Servlets* son:

1.Funciones.java	2.Historial.java	3.Login.java	4.Logout.java
5.UsuarioAU.java	6.UsuarioBD.java	7.UsuarioBR.java	8.UsuarioGD.java
9.UsuariosRegis.java	10.VigilanteBD.java	11.VigilanteHistorial.java	
12.ZonaBD.java	13.ZonaGD.java	14.Zonas.java	

CAPÍTULO 6

VALIDACIÓN

Es importante realizar una fase de pruebas una vez finalizado el desarrollo e implementación del sistema. Comprobaremos las prestaciones del proyecto y el funcionamiento deseado. Nos ayudará a garantizar el cumplimiento de los requisitos impuestos y encontrar posibles errores en las aplicaciones.

6.1. PRUEBAS BÁSICAS

Comenzaremos con las pruebas para verificar el funcionamiento de las diferentes comunicaciones de las que consta el proyecto.

6.1.1. LECTURA Y ESCRITURA DE DATOS

- LA TARJETA SEGURA Y CONTENEMOS LAS CLAVES NECESARIAS:

Contenemos las claves necesarias para leer y escribir en la tarjeta. La aplicación **AccessSecureTag**, nos permite realizar estas operaciones sobre las tarjetas, para comprobar el correcto funcionamiento.

-Escritura de Datos:

Introducimos los datos del empleado y pulsamos el botón “Aceptar”. Acercamos la tarjeta del empleado y los datos se guardan correctamente.

Figura 6.1: Escritura de Datos en la Tarjeta *Mifare* con claves

En la aplicación **Empleado**, el funcionamiento para guardar los datos es el mismo que el descrito anteriormente, con la diferencia de que no se tendrá que introducir ningún dato, porque esos datos, se los habrá pasado previamente el dispositivo “Vigilante”.

-Lectura de Datos:

Acercamos la tarjeta al dispositivo y cuando nos detecte la tarjeta, pulsamos el botón "Aceptar". Nos muestra correctamente los datos de la tarjeta.

Figura 6.2: Lectura de Datos en la Tarjeta *Mifare* con claves

- LA TARJETA SEGURA Y NO CONTENEMOS LAS CLAVES NECESARIAS:

Si no tenemos las claves correctas, no se podrán leer los datos ni escribir o modificar los datos en la tarjeta. Nos informará de que no se puede acceder, porque la tarjeta tiene seguridad.

Tanto en la escritura como en la lectura nos muestra por pantalla un mensaje de error.

'Tarjeta con seguridad, no se puede escribir' o 'Tarjeta con seguridad, no se puede leer'.

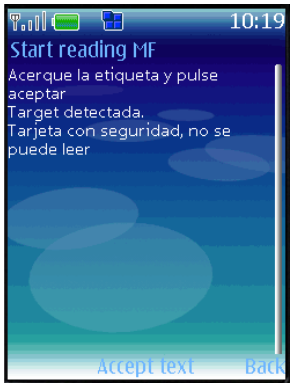


Figura 6.3: Escritura (a) y Lectura (b) en las Tarjetas *Mifare* sin claves

- LA TARJETA NO CONTIENE DATOS:

La lectura de datos de tarjetas, se realiza tanto con la aplicación **Vigilante** para leer datos de la tarjeta de un empleado, como por la aplicación **LectorFijo** que leerá los datos de la tarjeta del empleado que quiera acceder a la zona.

Si se lee alguna tarjeta que no contiene información guardada de datos personales, nos aparecerá un mensaje en la pantalla en ambas aplicaciones comunicándonos que no hay datos guardados en la tarjeta.

Figura 6.4: Lectura sin Datos en la Tarjeta (a) Vigilante (b) Lector Fijo

6.1.2. COMUNICACIÓN CON EL SERVIDOR BLUETOOTH

- EL USUARIO ESTÁ REGISTRADO EN LA BBDD:

Las aplicaciones **Vigilante** y **LectorFijo** se comunican con el servidor Bluetooth, para obtener los datos y zonas del empleado. Como vimos en el capítulo de Diseño, estas aplicaciones le envía al servidor el UID de la tarjeta del empleado que se quiere consultar. El servidor realiza una búsqueda en la BBDD, y si existe un usuario registrado con ese UID, se enviaran los datos pedidos.

- Aplicación Vigilante:

Si el usuario esta registrado obtendremos las zonas de acceso, y los datos personales del empleado. En pantalla se mostraran primero las zonas de acceso y a continuación una comparativa de los datos que el empleado tiene guardados en su tarjeta y los que hay en la BBDD referente a ese empleado.

Figura 6.5: Comunicación Bluetooth, Aplicación Vigilante

- Aplicación LectorFijo:

El Lector cuando recibe los datos comprueba que la zona a la que da acceso está contenida en las zonas a las que el empleado tiene acceso. Si es así, permitirá el paso del empleado, en caso contrario se le denegara el acceso.

Figura 6.6: Comunicación Bluetooth, Aplicación Lector Fijo

- EL USUARIO NO ESTA REGISTRADO EN LA BBDD:

Cuando el empleado del que solicitamos información, no está registrado en la BBDD, tanto en la aplicación **Vigilante** como en **LectorFijo**, nos aparecerá un mensaje indicándonos que ese empleado no está registrado.

Figura 6.7: Usuario no registrado en la BBDD

6.1.3. COMUNICACIÓN NFCIP

- COMUNICACIÓN CORRECTA, INTERCAMBIO DE DATOS.

- Dispositivo Vigilante:

Este dispositivo, se establece como el *Initiator* de la comunicación NFCIP. Será el que inicie la comunicación cuando encuentre al otro dispositivo.

La comunicación se establece correctamente, y al final de la pantalla nos indica que ésta ha finalizado.

-Dispositivo Empleado:

Este dispositivo, actúa como *Target* de la comunicación NFCIP. Esperara a recibir algún dato para poder iniciar el envió de información.

La comunicación se establece correctamente, y al final de la pantalla nos indica que ha finalizado, junto con los datos recibidos.

- COMUNICACIÓN CORTADA, NUEVO INTENTO:

Si los dispositivos se separan antes de la transmisión de todos los datos, la comunicación no habrá finalizado. Las conexiones NFCIP se cierran y se podrá volver a enviar los datos de nuevo, pulsando en ambos dispositivos de nuevo el botón 'Iniciar'.

6.2. PRUEBAS PRÁCTICAS

Una vez realizadas estas pruebas, comenzaremos a verificar el funcionamiento de las diferentes aplicaciones y la relación entre ellas.

6.2.1. CAMBIAMOS DATOS PERSONALES DEL EMPLEADO

Desde la página Web de gestión de los empleados, cambiamos los teléfonos de un empleado. El nuevo teléfono móvil es 658934710 y el teléfono fijo es 914562987. El teléfono móvil que tiene el empleado guardado en su tarjeta es 678967545 y el teléfono fijo 916543987.

Realizamos el cambio satisfactoriamente en la página web, y con la aplicación **Vigilante**, comprobamos los datos del empleado. Nos aparece esta imagen. Hemos comprobado el correcto funcionamiento.

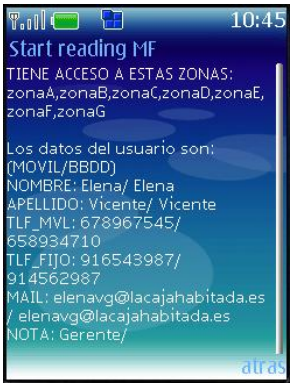


Figura 6.8: Datos del Empleado

En la imagen nos aparece los datos que tiene el empleado en la tarjeta, y los datos que hay en la BBDD. Vemos que el teléfono móvil y el teléfono fijo no coinciden.

El siguiente paso será pasarle los nuevos datos al empleado para que los modifique y tenga los datos correctos. Ponemos ambos móviles en la comunicación NFCIP y comenzamos la comunicación.

La comunicación finaliza correctamente. La aplicación **Empleado** guarda esa información en la tarjeta. Acercamos la tarjeta y los datos se guardan correctamente.

Para comprobar, leemos los nuevos datos de la tarjeta y comprobamos que contiene los datos correctos.

Figura 6.9: Lectura de Datos

6.2.2. CAMBIAMOS ZONAS DE ACCESO

Desde la página Web de gestión de empleados, cambiamos las zonas de acceso de un empleado. Le quitamos el acceso a la zonaA. El cambio lo realizamos correctamente en la página Web.

-Aplicación LectorFijo:

Con los nuevos permisos el empleado no podrá entrar en la zonaA. Para comprobar el correcto funcionamiento acercamos la tarjeta del empleado al Lector de la zonaA. El lector comprueba las zonas de acceso y le muestra correctamente el mensaje “NO TIENE ACCESO A LA ZONA”.

-Aplicación Vigilante:

En la aplicación **Vigilante** al comprobar los datos del empleado, no nos aparecerá la zonaA en la lista de zonas a las que tiene acceso. Funciona correctamente también.

6.2.3. HISTORIAL DEL VIGILANTE

Cada vez que el empleado Vigilante entra y sale de una zona se registra en el historial. Se crea un historial distinto cada día.

Comprobamos que no hay historial para el día actual. El empleado Vigilante entra en una zona.

Se crea correctamente el historial del día.

ID_HISTORIAL	FECHA
abcd1	(2011-03-03)
abcd2	(2011-03-04)
abcd5	(2011-04-30)
abcd6	(2011-05-02)
abcd7	(2011-05-12)
abcd8	(2011-05-14)

ID_HISTORIAL	FECHA
abcd1	(2011-03-03)
abcd2	(2011-03-04)
abcd5	(2011-04-30)
abcd6	(2011-05-02)
abcd7	(2011-05-12)
abcd8	(2011-05-14)
abcd9	(2011-05-25)

Tabla 6.1: Creación de Historial

Comprobamos que en ese historial hay una entrada con la zona a la que ese vigilante ha accedido. Esta entrada tendrá la hora de entrada a la zona. Comprobamos que es correcto. Salimos de la zona y comprobamos que la hora de salida se guarda correctamente también.

ID_HISTORIAL	ID_ZONA	ENTRADA	SALIDA
abcd9	zonaA	11:41:43	00:00:00

ID_HISTORIAL	ID_ZONA	ENTRADA	SALIDA
abcd9	zonaA	11:41:43	11:49:16

Tabla 6.2: Vista del Historial Actualizado

6.2.4. BORRAMOS UN EMPLEADO

Borramos correctamente a un empleado de la BBDD desde la página Web que gestiona los empleados.

Intentamos acceder a una zona. El lector le pide los datos al servidor Bluetooth. Le comunica que no hay empleado registrado con esos datos. Y en la pantalla del Lector nos aparecerá el mismo mensaje que mostramos anteriormente cuando no había empleado registrado. Por lo tanto comprobamos el correcto funcionamiento.

El vigilante al comprobar los datos de ese empleado, recibe del servidor Bluetooth la misma información que el Lector, y muestra el mismo mensaje por pantalla.

6.2.5. AÑADIMOS UN EMPLEADO

Añadimos mediante la página Web un nuevo empleado correctamente.

Intentamos acceder a una zona, el Lector pide los datos al servidor Bluetooth, que comprueba que el usuario existe. Le envía las zonas a las que tiene acceso. El Lector le permite el paso si tiene acceso a su zona.

El vigilante comprueba los datos de ese empleado, comprueba que el empleado esta registrado en la BBDD. Funcionamiento correcto comprobado.

6.3. CONCLUSIONES DEL CAPITULO

En este capítulo se han realizado una serie de pruebas para obtener resultados que nos permitan validar la propuesta planteada. Para ello se han analizado cada una de las diferentes partes de que consta el proyecto por separado, para verificar el correcto funcionamiento. Después se probó el funcionamiento global de la aplicación, verificando una vez su funcionamiento correcto.

Con todo esto se ha podido ver que las medidas realizadas cumplen ampliamente con las expectativas, ya que se ha conseguido darle seguridad a las tarjetas *Mifare* 4K, guardar y leer los datos de forma segura, comunicar dos dispositivos para el intercambio de datos mediante la tecnología NFC, conexión Bluetooth con el servidor y seguimiento de las rondas que realizan los vigilantes.

Podemos afirmar con todo esto, que la propuesta realizada queda validada para los objetivos que se habían marcado.

CAPÍTULO 7

CONCLUSIONES Y TRABAJOS FUTUROS

7.1. CONCLUSIONES

En el mercado existen aplicaciones que controlan el acceso a zonas, y gestionan la identidad de los usuarios. En este proyecto se han implementado diferentes aplicaciones para que toda esta funcionalidad pueda ser ejecutada en dispositivos móviles.

Hemos utilizado dos tecnologías inalámbricas, Bluetooth y NFC, que resultan interesantes para aplicaciones dirigidas a funcionar en dispositivos móviles, por las posibilidades de implementación que ofrecen. Además son tecnologías costeables, y perfectamente pueden convivir con las tecnologías existentes.

La integración de NFC y J2ME es una buena combinación, y muchos fabricantes de teléfonos móviles están descubriendo el gran potencial de utilizar la tecnología NFC en estos dispositivos[26]. Se ha podido comprobar que la información sobre esta tecnología ha crecido notablemente a lo largo del año de duración del proyecto. Se han comenzado a realizar proyectos pilotos en entornos reales a gran escala, lo que indica que puede llegar a consolidarse con el paso de los años.

El desarrollo y la implementación del proyecto nos ha permitido comprobar todas las ventajas de la tecnología NFC, y la gran usabilidad que tienen a día de hoy todas estas tecnologías.

Además con las pruebas realizadas se ha comprobado el correcto funcionamiento del sistema completo. Resultando muy interesante la posibilidad de integrar todas las tecnologías.

[26] Google Wallet, http://es.engadget.com/tag/GoogleWallet/

Una vez finalizado su desarrollo e implementación, y realizada las respectivas pruebas vemos las ventajas e inconvenientes que este sistema lleva. Podremos proponer mejoras futuras que solucionen los inconvenientes, como veremos a continuación.

Por otro lado vemos que es una tecnología fácil de utilizar y sencilla, no requiere de conocimientos técnicos altos para su uso. Además, integrada en los dispositivos móviles nos permite tener menús claros y sencillos que facilitan su uso.

7.2. TRABAJOS FUTUROS

Aunque el desarrollo del proyecto es bueno, existen puntos importantes a desarrollar que harán que estas aplicaciones sean más prácticas y puedan emplearse en un entorno real.

- Disponer de un Lector NFC, en el módulo lector fijo, en vez de un móvil Nokia 6131. Implementar la funcionalidad deseada en un Lector NFC. Esto nos permitirá comunicar este Lector con una puerta electrónica, que se abrirá sólo si tenemos permisos.
- Comunicación por cable entre el Lector NFC y el Servidor. Una vez implementado el Lector NFC, se podrá conectar por cable al servidor, para que las consultas a la BBDD se realice de una forma rápida y segura.
- Conexión Wifi en vez de conexión Bluetooth. Los dispositivos Nokia 6131 no nos permiten utilizar la conexión Wifi para la realización de consultas y transmisión de información. Una mejora seria cambiar la conexión Bluetooth por conexión Wifi, o tener la posibilidad de poder utilizar las dos, por si alguna falla en algún momento.
- Disponer de las firmas necesarias, para que las aplicaciones corran en el dominio de seguridad del dispositivo y así tener los datos en la etiqueta interna del elemento seguro.
- Desarrollo de todas estas aplicaciones para otros sistemas operativos como *Android*, ya que la actualidad de los móviles con la tecnología NFC tienen este sistema operativo.
- Mejorar el sistema completo para poder desplegarlo en un entorno turístico real, o adaptarlo a otros entornos, como empresas, universidades ...

- Ampliar las funcionalidades del sistema a los usuarios que se hospedan en el Hotel, pudiendo tener llaves de acceso a las habitaciones en los dispositivos móviles, o implementando *tags* que permitan activar la configuración para conectarse a la Wifi, o descargarse planos de la ciudad u otro tipo de información turística.

CAPÍTULO 8

PRESUPUESTO

En este capítulo se detallan los aspectos económicos realizando un análisis del desarrollo de este proyecto.

8.1. TAREAS

En este apartado, se muestra una estimación de la duración de las distintas tareas realizadas para la finalización del proyecto. Estas tareas no se han realizado de forma continua debido a que el proyecto se ha alternado con estudios, becas y trabajo.

El comienzo del proyecto se desarrollo bajo la dirección de la empresa "Gamma Solutions" pero finalmente se decidió cambiar gran parte del diseño y la implementación, modificando y añadiendo otras funciones, proporcionando una mayor funcionalidad a todo el sistema.

Esto nos permite mostrar una aproximación de la duración del proyecto. En la tabla aparecen las diferentes fases junto con el número de horas estimadas para su realización.

FASE	DESCRIPCIÓN	NÚMERO DE HORAS
1	Investigación y documentación: NFC y Bluetooth	60 horas
2	Despliegue y configuración del entorno del trabajo	12 horas
3	Toma de contacto con la tecnología NFC y Bluetooth	60 horas
4	Ideas y diseño de la aplicación	80 horas
5	Desarrollo funcionalidades NFC	160 horas
6	Desarrollo funcionalidades Bluetooth	90 horas
7	Integración funcionalidades NFC y Bluetooth	100 horas
8	Diseño y desarrollo de la base de datos	40 horas
9	Desarrollo de la aplicación Web	50 horas
10	Integración del sistema completo	60 horas
11	Pruebas del sistema	40 horas
12	Redacción de la memoria	100 horas
	TOTAL HORAS	**852**

Tabla 8.1: Tareas del proyecto

8.2. COSTES

En este apartado se detallan los costes necesarios para realización del proyecto. Estos costes son debidos a diferentes factores, personal, material (equipos, componentes, software empleado y licencias) y costos indirectos.

8.2.1. PERSONAL

Para la realización del proyecto se ha necesitado cubrir las tareas que realizarían los especialistas que se detallan en la tabla 7.2, junto con el precio de mano de obra por hora. El jefe del proyecto sería el tutor.

CARGO	COSTE (POR HORAS)
Analista	35 €
Diseñador	40 €
Gestión de Calidad y Pruebas	35 €
Jefe de proyecto	45 €
Programador	25 €

Tabla 8.2: Costes de Personal

Si estos costes se aplican al número de horas dedicado por cada especialista en función de las fases del diseño y las tareas encomendadas, se calcula el coste total del personal requerido para este proyecto, a través de la siguiente la tabla 7.3.

	FASE	CARGO	NÚMERO DE HORAS	COSTE
1	Investigación	Analista	60 horas	2100 €
2	Entorno de trabajo	Programador	12 horas	300 €
3	Diseño del sistema	Diseñador	60 horas	2400 €
4	Toma de contacto	Programador	80 horas	2000 €
5	Desarrollo NFC	Programador	160 horas	4000 €
6	Desarrollo Bluetooth	Programador	90 horas	2250 €
7	Integración	Programador	100 horas	2500 €
8	Desarrollo BBDD	Programador	40 horas	1000 €
9	Desarrollo página web	Programador	50 horas	1250 €
10	Integración sistema	Programador/Diseñador	60 horas	1800 €
11	Pruebas	Gestión de Calidad y pruebas	40 horas	1400 €
12	Redacción memoria	Análisis/Diseñador	100 horas	3750 €
			TOTAL COSTE	**24 750 €**

Tabla 8.3: Costes Totales de Personal

8.2.2. MATERIAL

En este apartado se recoge el coste de los materiales empleados durante la realización del proyecto. Estos costes se pueden dividir en equipo y licencias.

EQUIPO

En la tabla 7.4 se detallan los componentes utilizados y su coste.

DISPOSITIVO	COSTE/UNIDAD	UNIDADES	COSTE TOTAL
Nokia 6131 NFC	190 €	1	190 €
Tarjetas *Mifare* 4K	0,90 €	6	5,4 €
Ordenador Portátil	680 €	1	680 €
Dispositivo USB Bluetooth	12 €	1	12 €
		COSTE TOTAL	**887,4 €**

Tabla 8.4: Costes Equipo

LICENCIAS

Mostramos en la tabla 7.5 las licencias utilizadas.

CONCEPTO	PRECIO/UNIDAD	UNIDADES	PRECIO TOTAL
JAVA SE Development Kit	0 €	1	0 €
Nokia SDK 6131 NFC	0 €	1	0 €
Netbeans IDE 6.9	0 €	1	0 €
Librería Bluecove 2.1.0	0 €	1	0 €

Tabla 8.5: Costes Licencias

8.2.3. INDIRECTOS

En este apartado se resumen los costes, derivados del uso de instalaciones para la realización del proyecto. Se toma como tiempo estimado de realización del proyecto un año.

CONCEPTO	COSTE
Alquiler del local	1440 €
Luz y agua	960 €
Teléfono y conexión a Internet	600 €

Tabla 8.6: Costes Indirectos

8.3 TOTAL

Teniendo en cuenta todos los costes relatados anteriormente, el coste total necesario para realizar el proyecto asciende a 28.637,4 €, tal y como se muestra en la tabla 7.7.

CONCEPTO	COSTE
Personal	24750 €
Equipo	887,4 €
Licencias	0 €
Costes Indirectos	3000 €
COSTE TOTAL	**28637,4 €**

Tabla 8.7: Resumen de Costes

Esta cantidad no tiene en cuenta ni el análisis del riesgo ni los impuestos. Por ello en la tabla 7.8 se desglosa el balance final del proyecto.

CONCEPTO	COSTE
COSTE TOTAL	28637,4 €
RIESGO (10%)	2863,74 €
BENEFICIO (20%)	5727,48 €
TOTAL SIN I.V.A	37228,62 €
I.V.A (18%)	6701,15 €
TOTAL COSTE	**43929,77 €**

Tabla 8.8: Presupuesto Total

Teniendo en cuenta los costes desglosados en los apartados anteriores, el presupuesto total de este proyecto asciende a la cantidad de CUARENTA Y TRES MIL NOVECIENTOS VEINTINUEVE CON SETENTA Y SIETE euros.

Leganés, a de Junio de 2011

La ingeniera proyectista

Fdo: Elena Vicente García

GLOSARIO

AES	Advanced Encryption Standard
APDU	Application Protocol Data Unit
API	Application programming interface.
BCC	Block Check Character
CLDC	Connected Limited Device Configuration
EDR	Enhanced Data Rate
GNU GPL	General Public License
HCI	Host Controller Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IEC	International Electrotechnical Commission
IEEE	Institute Of Electrical And Electronics Engineers
IDE	Integrated Development Environment
ISM	Industrial Scientific and Medical (band)
ISO	International Standard Organization
J2EE	Java 2 Platform Enterprise Edition
J2ME	Java 2 Platform Micro Edition
J2SE	Java 2 Platform Standard Edition
JDBC	Java Database Connectivity
JDK	Java Development Kit
JRE	Java Runtime Environment
JSP	Java Server Pages
JSR	Java Specification Request
L2CAP	Logical Link Control and Adaptation Protocol
MAC	Media Access Control

MAD	Mifare Application Directory
MIDP	Mobile Information Device Profile
MMC	Multi Media Card
MNO	Mobile Network Operator
MS	Memory Stick
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NFCIP	Near Field Communication Interface and Protocol
P2P	Peer to Peer
PDA	Personal Digital Assistant
PHP	Hypertext Preprocessor
PIN	Personal Identification Number
RAM	Random Access Memory
RF	Radio Frecuency
RFCOMM	Radio Frecuency Communication
RFID	Radio Frecuency Identification
RTD	Record Type Definition
SD	Secure Digital
SDK	Software Development Kit
SIG	Special Interest Group
SIM	Subscriber Identification Module
SMS	Short Message Service
SPP	Serial Port Profile
SQL	Structured query language
SWP	Single Wire Protocol
TSM	Trusted Service Manager
UICC	Universal Integrated Circuit Card
UID	Unique Identifier

UIE	Unified Emulator Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
Wifi	Wireless Fidelity
WPAN	Wireless Personal Area Network
XML	Extensible Markup Language

BIBLIOGRAFÍA

[1] Xataka, “2011, el año de la tecnología NFC”, 25 Enero 2011. *http://www.xatakamovil.com/conectividad/2011-el-ano-de-la-tecnologia-nfc* [Última vista, Enero 2011]

[2] Terra, “¿Qué es NFC?” ,2008. *http://www.terra.es/personal/ccossio/tecnologiaNFC_2.htm*

[Última vista, Octubre 2010]

[3] Wikipedia.org, *http://en.wikipedia.org/wiki/Near_field_communication* [Última vista, Febrero 2010]

[4] NFC Forum, “NFC Tecnhology Overview”, Septiembre 2009. *http://www.nfc-forum.org/events/oulu_spotlight/Technical_Architecture.pdf* [Última vista, Noviembre 2010]

[5] NFC Forum,”Especificaciones técnicas” *http://www.nfc-forum.org/specs/spec_list/#protts* [Última vista, Abril 2009]

[6] Mami.ucle.es*, http://mami.uclm.es/nuevomami/publicaciones/UCAmI-chavira.pdf* [Última vista, Febrero 2010]

[7] Telefónica España, 8 Julio 2008. *http://www.bcndigital.org/Lists/BDIGITAL%20Debates/Attachments/96/presentacio_AngelDavidGarcia.pdf* [Última vista, Mayo 2011]

[8] bdigital.eafit.edu.co “Marco Teórico”. *http://bdigital.eafit.edu.co/bdigital/PROYECTO/P621.3845CDM828/marcoTeorico.pdf* [Última vista, Mayo 2011]

[9] worlditsdirectory.com, Mayo 2008.“Dispositivos NFC para validación en Transporte Público”. *http://www.worlditsdirectory.com/ftp/080521%20:.%20JORNADA%20ITS%20EN%20ASTURIAS,%20OVIEDO/SESION%202-ITS%20EN%20EL%20TRANSPORTE%20PUBLICO/ANTONIO%20CANDEL,%20INDRA.pdf* [Última vista, Mayo 2011]

[10] Emt Málaga, 2008. “Pago por móvil NFC” *[http://www.emtmalaga.es/portal/page/portal/EMT/Pago%20por%20m%C3%B3vil%20NFC* [Última vista Mayo, 2011]

[11] Hotel Clarion. Mayo 2010.*http://www.clarionstockholm.com/nfc-project* [Última vista, Mayo 2011]

[12] Nexotur.com, Marzo 2011. *http://www.nexotur.com/nexobus/emt/madrid/anuncia/experiencia/piloto/pagar/autobus/urbano/traves/movil/37330/* [Última vista Junio, 2011]

[13] Engadget, Mayo 2011. *http://es.engadget.com/tag/GoogleWallet/* [Última vista, Mayo 2011]

[14] Itespresso.*es*”La banca europea se une en torno a los pagos NFC”, 6 Mayo 2011. *http://www.itespresso.es/la-banca-europea-se-une-en-torno-a-los-pagos-nfc-50863.html* [Última Vista, 1 Junio 2011].

[15] ActualidadGaget.com, Abril 2011. *http://www.actualidadgadget.com/2011/04/13/tres-telefonos-samsung-bada-con-tecnologia-nfc/* [Última vista, Mayo 2011]

[16] SIG Bluetooth, *http://www.bluetooth.com/Pages/Bluetooth-Home.aspx* [Última vista, Noviembre 2010]

[17] Todomonografias, *http://www.todomonografias.com/telecomunicaciones/bluetooth/* [Última vista, Febrero 2011]

[18] PaloWireless, Febrero 2001. *http://www.palowireless.com/bluearticles/cc1_security1.asp* [Última vista Noviembre 2010]

[19] Iec.csic.es, *http://www.iec.csic.es/criptonomicon/java/quesjava.html*. [Última visita Noviembre 2010]

[20] Icc.uma.es, *http://www.lcc.uma.es/~galvez/J2ME.html*. [Última visita Noviembre 2010]

[21] Programacion.com, *http://www.programacion.com/articulo/servlets_y_jsp_82* [Última visita Diciembre 2010]

[22] Jesús Arias Fisteus, Uc3m, “Desarrollo de aplicaciones Web con Servlets Y JSPs”, 2009. http://www.it.uc3m.es/labttlat/material/integracion.pdf. [Última visita Diciembre 2010]

[23] Programacion.com, *http://www.programacion.com/articulo/tomcat_-_introduccion_134* [Última vista, Octubre 2010]

[24] Proyecto de Luis Álvarez Álvarez, “Diseño y desarrollo de una aplicación Web para gestión docente”, Abril 2009. [Última vista, Octubre 2010]

[25] Wiki Forum Nokia, Mayo 2011. *http://wiki.forum.nokia.com/index.php/Nokia_6131_NFC_-_FAQs* [Última vista Febrero 2011]

[26] “Nokia 6131 NFC SDK Programmers Guide v1”, Julio 2003. *http://www.forum.nokia.com* [Última vista Marzo 2011]

[27] “Practical Attacks on the Mifare Classic by Wee Hon Tan”, Septiembre 2009. *http://www.doc.ic.ac.uk/~mgv98/MIFARE_files/report.pdf* [Última Vista 26 Febrero 2011].

[28] “NXP Type MF1K/4K Tag Operation”, Agosto 2007. *http://www.nxp.com/documents/application_note/AN130411.pdf* [Última Vista 26 Febrero 2011]

[29] library.forum.nokia.com “Nokia Extension for JSR 257”, *http://library.forum.nokia.com/index.jsp?topic=/Java_Developers_Library/GUID-D38F02EB-1628-489B-997D-EF723C16E3CF/overview-summary.html*, [Última vista Marzo 2010]

[30] Blucove.org, *http://bluecove.org/bluecove/apidocs/index.html*. [Última visita, Marzo 2011]

ANEXO A: GUIA DE INSTALACIÓN DEL SOFTWARE

Se detallan en el anexo los detalles de la instalación y configuración del software necesario para el desarrollo del proyecto.

Java SE Development Kit

La instalación del JDK (Java Development Kit) es necesaria, puesto que en él reside el JRE, e incluye herramientas como el compilador de Java, *Javadoc* para generar documentación o el depurador.

El archivo se puede descargar desde el enlace: *http://java.sun.com/products/archive/j2se/6u16/index.html*

La instalación es sencilla, solo hay que seguir los pasos que la propia ejecución genera, y seleccionar los distintos elementos que queremos instalar. Además la actualización es automática.

Neatbeans IDE 6.9

Este entorno de trabajo proporciona todas las herramientas con las que poder trabajar.

La descarga se realiza desde el enlace: *http://netbeans.org/downloads/6.9/*

Se deberá de descargar la versión “All”, y se obtendrá un archivo “.exe”, que al ejecutarlo, mostrara una serie de pasos para la configuración de la instalación.

Con este entorno de trabajo realizaremos las aplicaciones Java SE, Java ME y Java Web, además de desplegar el servidor web, con las herramientas ofrecidas por el IDE como Apache Tomcat 6.0.26.

Netbeans IDE 6.9 funciona sobre una *Java SE Development Kit (JDK)*, por lo tanto, su instalación previa es requerida. Concretamente, se precisa la versión JDK 6 *Update* 13 o posterior.

Apache Tomcat 6.0.26

El servidor Apache Tomcat se instala a partir de la instalación de Eclipse IDE. Para su uso, no es necesaria ninguna configuración. La propia aplicación Web GestionEmpleados, al ser ejecutada, se encarga de arrancar el servidor.

Nokia 6131 NFC SDK 1.1

Este SDK es propietario de Nokia. Es posible la integración con Netbeans para tener una plataforma única de desarrollo.

En el apartado 3.1 del documento, se describen las características de este SDK.

La descarga se realiza desde el enlace: *http://www.forum.nokia.com/info/sw.nokia.com/id/ef4e1bc9-d220-400c-a41d-b3d56349e984/Nokia_6131_NFC_SDK.html*

En este enlace, hay una guía de instalación y una guía del programador.

Para integrarlo con Netbeans se deben seguir los siguientes pasos:

En Netbeans, se hace *click* en "*Herramientas->Plataformas Java*".

Allí se selecciona "*Añadir Plataforma*".

Se elige "*Custom Java ME MIDP Plataform Emulator*".

En el siguiente paso, se selecciona en nuestro sistema de archivos, la carpeta donde se ha instalado el SDK Nokia.

Librería Bluecove 2.1.0

Esta librería se detalla en el apartado 3.2. Basada en la implementación JSR-82, para plataforma Java.

Se ha de añadir en el proyecto de la aplicación del servidor Bluetooth, para posibilitar su comunicación con otro dispositivo. Se ha utilizado para este proyecto la versión Bluecove 2.1.0.

El enlace de descarga es: *http://bluecove.org/*

Cabe destacar que en la configuración de la aplicación NFC Cliente en Netbeans, se deberá elegir: Nokia_6131_NFC_SDK_1_1, MIDP-2.0, CLDC-1.1 y el API opcional *lib/ext/nfc.zip* y JSR82-1.0 (que nos permite la comunicación Bluetooth).

MySQL Server

Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario.

La descarga la realizamos desde el enlace: *http://www.mysql.com/downloads/mysql-5.0.html*

Su instalación es sencilla, solo hay que seguir los pasos que la propia ejecución genera.

MySQL Connectors J

Es un Driver que nos permite utilizar MySQL desde Netbeans debemos descargarnos el conector necesario. El conector J es el controlador de base de datos estandarizado para plataformas java.

La descarga se realiza desde el enlace: *http://dev.mysql.com/downloads/connector/j/*

Se ha de añadir en el proyecto de la aplicación del servidor GestionEmpleados, para posibilitar su comunicación con la base de datos.

MySQL Workbench 5.2 CE

Es una herramienta de modelado de bases de datos visual multiplataforma, desarrollada por MySQL. Permite generar el código SQL a partir del modelado de datos creado y viceversa.

La descarga se realiza desde el enlace: *http://dev.mysql.com/downloads/workbench/*

Para su instalación únicamente hay que seguir los pasos que la propia ejecución genera. Nos permite generar los scripts necesarios para crear la base de datos.

ANEXO B: MANUAL DE USUARIO

B.1. APLICACIÓN SECUREWRITERMF

Esta aplicación es muy sencilla, únicamente hay que acercar la tarjeta y las claves se modifican.

B.2. APLICACIÓN ACCESSSECURETAG

Esta aplicación contiene cuatro opciones. Dos para leer y escribir en la etiqueta interna del dispositivo y dos para leer y escribir en una tarjeta externa.

Para grabar los datos de un empleado en una tarjeta externa *Mifare*, seleccionaremos la opción “Escribir Externo”.

La aplicación muestra los campos que debemos rellenar. Nombre, Apellido, Tlf móvil, Tlf Fijo, Mail y Nota.

Una vez rellenados los campos pulsaremos el botón “Aceptar”. La aplicación nos muestra los datos que queremos grabar para comprobar que son correctos. Si hubiese algún error pulsaríamos la tecla “back” y volveríamos a la pantalla anterior.

Si los datos son correctos solo debemos acercar la tarjeta para que se graben los datos. Una vez guardados la aplicación muestra el mensaje ‘Hemos guardado los datos del contacto en la tarjeta’.

B.3. MÓDULO EMPLEADO

Esta aplicación informa al arrancarla de que es el dispositivo *Target* y se debe pulsa el botón “Iniciar”, para establecer una comunicación *peer to peer*.

Una vez pulsado “Iniciar” muestra en la pantalla el mensaje ‘Buscando Initiator…’. En cuanto detecta el dispositivo *Initiator* se produce el intercambio de mensajes, y para comprobar que la comunicación ha finalizado correctamente la aplicación muestra el mensaje ‘Comunicación finalizada’.

Una vez finalizada la comunicación se pulsa el botón “Opciones” y mostrará un menú con las cuatro opciones de la aplicación anterior, se selecciona la opción “Escribir Externo” y nos indicará los datos que se deben guardar, como en la aplicación anterior.

B.4. MÓDULO VIGILANTE

La aplicación de este modulo lee los datos de las etiquetas de los empleados. Cuando se detecta una tarjeta se pulsa el botón “Aceptar” y a continuación se intenta establecer comunicación con algún servidor Bluetooth de la red del Hotel. Una vez establecida la comunicación se mostrara por pantalla el resultado de la consulta.

Se mostraran las zonas a las que ese empleado tiene acceso, así como los datos leídos de la tarjeta y obtenidos de la BBDD para compararlos.

Se pulsa el botón “atrás” tanto si se quiere continuar con la comunicación NFC, para seguir leyendo tarjetas de empleados en cuyo caso se pulsa el botón “Leer”, como si se quiere establecer la comunicación *peer to peer*, en cuyo caso se pulsa el botón “Escribir”.

Si se pulsa el botón “Escribir” comenzaremos la comunicación *peer to peer*. Para ello primero rellenamos los datos que se quieren enviar al otro dispositivo y después se pulsará el botón “Inciar”.

Cuando se pulsa ese botón, aparece un mensaje ‘Buscando Target…’. Cuando encuentra al dispositivo *Target* se produce el intercambio de mensaje. Y nos comunica que la comunicación ha finalizado. Se debe pulsar el botón `leer` para volver a la comunicación NFC de lectura de tarjetas.

B.5. MÓDULO LECTOR FIJO

La aplicación de este módulo cuando se arranca, se queda a la espera de que se detecte alguna tarjeta *Mifare*. Una vez detectada se comunica con el servidor Bluetooth y muestra por pantalla un mensaje dependiendo de si tiene acceso o no el empleado a la zona. No hay que configurar nada en esta aplicación, ni pulsar ningún botón.

B.6. MÓDULO SERVIDOR BLUETOTH

Hay que arrancar la aplicación en el Servidor. Se arranca directamente desde el Netbeans.

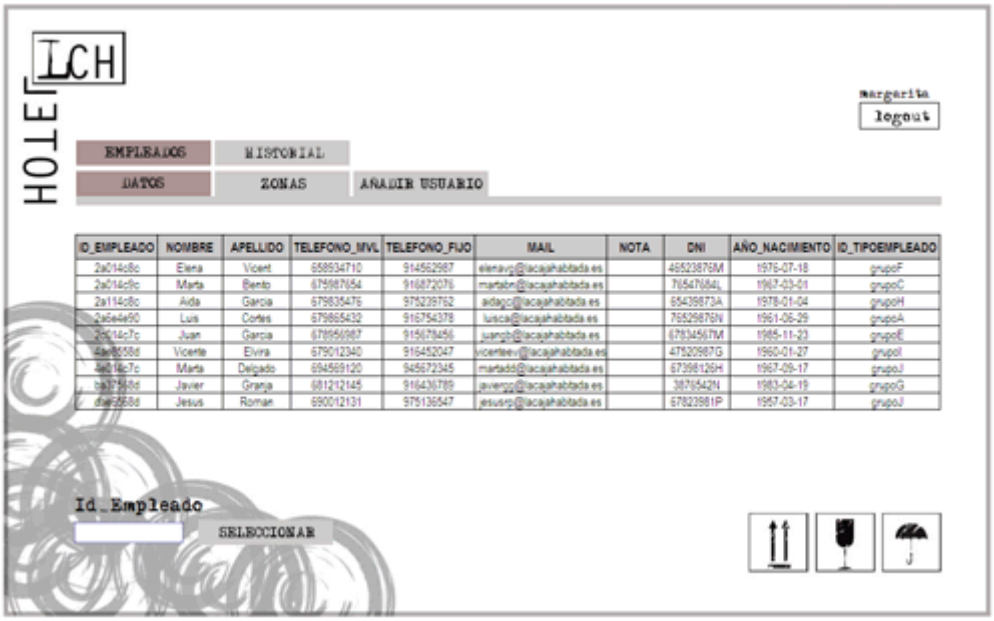
B.7. MÓDULO RED HOTEL

Aplicación Web GestionEmpleados

Desde la portada de la web, el empleado del Hotel se logea, metiendo su nombre y *password.*

Una vez logeados, nos aparecerá en la pantalla dos opciones a escoger. Empleados e Historial.

Si pulsamos la opción Empleados nos mostrara una tabla con todos los empleados registrados en la BBDD.



Se puede seleccionar un empleado introduciendo el Id_Empleado, para modificar sus datos, borrar el empleado o modificar sus zonas.

Hay una serie de pestañas en la parte de arriba de la pagina web. La pestaña “Empleados” muestra las opciones disponibles para gestionar empleados. Mostrar sus datos, mostrar sus zonas y añadir usuario. Estas opciones corresponden con las diferentes pestañas que aparecen debajo.

Tenemos la pestaña “Historial” que muestra los Vigilantes de seguridad. Se puede seleccionar uno para ver los historiales abiertos.